USER MANUAL

DMC-3425

Manual Rev. 1.1b

By Galil Motion Control, Inc.

Galil Motion Control, Inc.
3750 Atherton Road
Rocklin, California 95765
Phone: (916) 626-0101
Fax: (916) 626-0102
Internet Address: support@galilmc.com
URL: www.galilmc.com

Rev 6/06

Contents

Appendices 171

Index 199

Chapter 1 Overview

Introduction

The DMC-3425 provides a highly versatile, powerful form of distributed control where multiple DMC-3425 controllers can be linked together on the Ethernet. One DMC-3425 is designated as a "master" that receives all commands from the host computer and passes them to the other "slave" DMC-3425 controllers. Efficient, quick communications are realized as this approach eliminates the usual, multiple communication links between the host computer and each controller.

Each DMC-3425 precisely controls two servo motors, providing ECAM, gearing and both linear and circular interpolation for coordinated motion along the two local axis. A single axis DMC-3415 is also available. When acting as the "master," a DMC-3425 can receive PR, PA and JG commands for up to eight axes and distribute them to the appropriate controller. Coordinated motion is commanded locally by each DMC-3425 "slave" controller. Performance capability of these controllers includes: 12 MHz encoder input frequency, 16-bit motor command output DAC, +/-2 billion counts total travel per move, 250 μsec minimum sample rate and non-volatile memory for program and parameter storage. Designed for maximum flexibility, the DMC-3425 can be interfaced to a variety of motors and drives including step motors, brush and brushless servo motors and hydraulics. The DMC-3425 can also be configured to provide sinusoidal commutation for brushless motors.

The controller accepts feedback from a quadrature linear or rotary encoder with input frequencies up to 12 million quadrature counts per second. Modes of motion include jogging, point-to-point positioning, electronic cam, electronic gearing and contouring. Several motion parameters can be specified including acceleration and deceleration rates and slew speed. The DMC-3425 also provides motion smoothing to eliminate jerk.

For synchronization with outside events, the DMC-3425 provides uncommitted I/O. The DMC-3425 provides up to 3 digital inputs, 3 digital outputs and 2 analog inputs. The DMC-3415 provides 7 digital inputs, 3 digital outputs and 2 analog inputs. Committed digital inputs are provided for forward and reverse limits, abort, home, and definable input interrupts. An additional 64 configurable I/O points may be added with the optional DB-14064 daughter card. The DMC-3425 distributed system may also be linked with multiple IOC-7007 Ethernet I/O modules for complete machine I/O control. Event triggers can automatically check for elapsed time, distance and motion complete.

The DMC-3425 is easy to program. Instructions are represented by two letter commands such as BG for Begin and SP for Speed. Conditional instructions, Jump statements and arithmetic functions are included for writing self-contained applications programs. An internal editor allows programs to be quickly entered and edited, and support software such as the WSDK allows quick system set-up and tuning. Commands may also be sent in Binary to decrease processing time.

To prevent system damage during machine operation, the DMC-3425 provides many error-handling features. These include software and hardware limits, automatic shut-off on excessive error, abort input and user-definable error and limit routines.

The DMC-3425 is designed for stand-alone applications and provides non-volatile storage for programs, variables and array elements.

This manual uses 'DMC-3425' to refer to the distributed control E-series from Galil. However, most functions described in this manual are available using either the DMC-3425 or the DMC-3415. If a function is specific to only one of the controllers, this will be explicitly stated.

Overview of Motor Types

The DMC-3425 can provide the following types of motor control:

1. Standard servo motors with +/- 10 volt command signals
2. Step motors with step and direction signals
3. Brushless servo motors with sinusoidal commutation
4. Other actuators such as hydraulics - For more information, contact Galil.

The user can configure each axis for any combination of motor types, providing maximum flexibility.

Standard Servo Motors with +/- 10 Volt Command Signal

The DMC-3425 achieves superior precision through use of a 16-bit motor command output DAC and a sophisticated PID filter that features velocity and acceleration feedforward, an extra notch filter and integration limits.

The controller is configured by the factory for standard servo motor operation. In this configuration, the controller provides an analog signal (+/- 10Volt) to connect to a servo amplifier. This connection is described in Chapter 2.

Stepper Motor with Step and Direction Signals

The DMC-3425 can control 2 stepper motors. In this mode, the controller provides two signals to connect to each stepper motor: Step and Direction. For stepper motor operation, the controller does not require an encoder and operates the stepper motor in an open loop. Chapter 2 describes the proper connection and procedure for using stepper motors.

NOTE: In order to use two stepper motors on the DMC-3425, the controller must be ordered as a DMC-3425-Stepper. In this mode, the Amp Enable and Error outputs are converted to the Step and Direction signals for the Y-axis. Contact Galil for other stepper options.

Brushless Servo Motor with Sinusoidal Commutation

The DMC-3415 can provide sinusoidal commutation for brushless motors (BLM). In this configuration, the controller generates two sinusoidal signals for connection with amplifiers specifically designed for this purpose. Please note, for a 2 axis DMC-3425, converting to a brushless motor uses up the second axis.

Note: The task of generating sinusoidal commutation may be accomplished in the brushless motor amplifier. If the amplifier generates the sinusoidal commutation signals, only a single command signal is required and the controller should be configured for a standard servo motor (described above).

Sinusoidal commutation in the controller can be used with linear and rotary BLMs. However, the motor velocity should be limited such that a magnetic cycle lasts at least 6 milliseconds*. For faster motors, please contact the factory.

The controller provides a one-time, automatic set-up procedure. The parameters determined by this procedure can then be saved in non-volatile memory to be used whenever the system is powered on.

The DMC-3415 can control BLMs equipped with Hall sensors as well as without Hall sensors. If hall sensors are available, once the controller has been setup, the controller will estimate the commutation phase upon reset. This allows the motor to function immediately upon power up. The Hall effect sensors also provide a method for setting the precise commutation phase. Chapter 2 describes the proper connection and procedure for using sinusoidal commutation of brushless motors.

* 6 Milliseconds per magnetic cycle assumes a servo update of 1 msec (default rate).

DMC-3425 Functional Elements

The DMC-3425 circuitry can be divided into the following functional groups as shown in Figure 1.1 and discussed below.

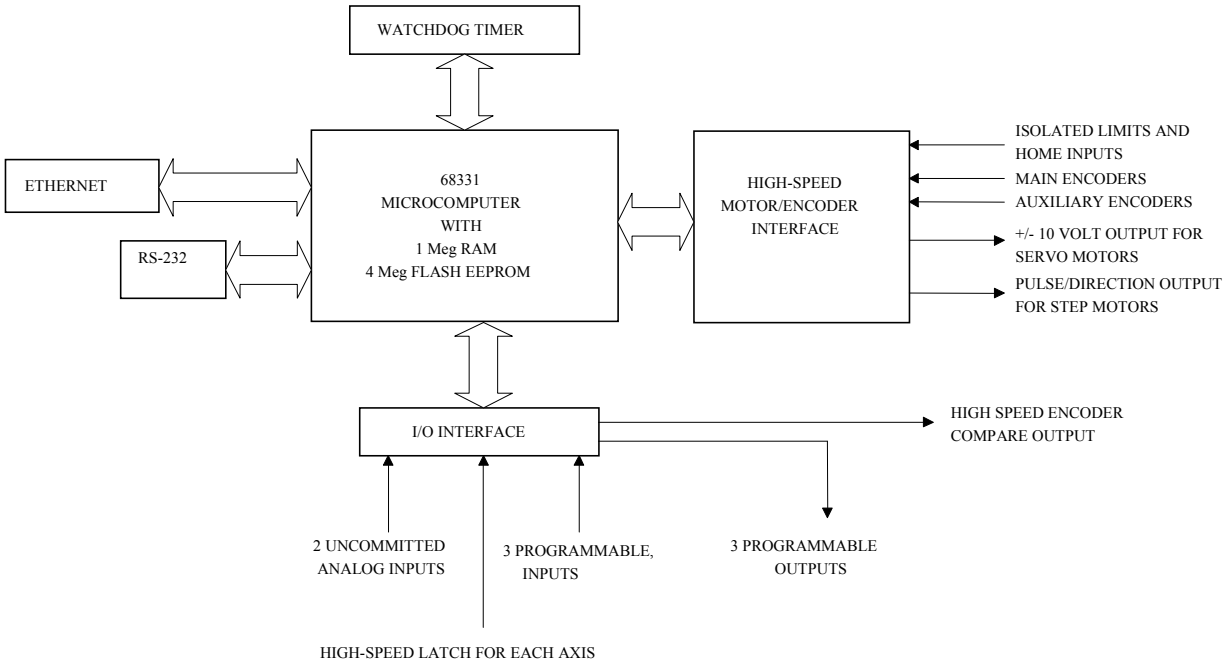


Figure 1.1 - DMC-3425 Functional Elements

Microcomputer Section

The main processing unit of the DMC-3425 is a specialized 32-bit Motorola 68331 Series Microcomputer with 1 Meg RAM and 4 Meg Flash EEPROM. The RAM provides memory for variables, array elements and application programs. The flash EEPROM provides non-volatile storage of variables, programs, and arrays. It also contains the DMC-3425 firmware.

Motor Interface

Galil's GL-1800 custom, sub-micron gate array performs quadrature decoding of each encoder at up to 12 MHz. For standard servo operation, the controller generates a +/-10 Volt analog signal (16 Bit DAC). For sinusoidal commutation operation, the controller uses two DACs to generate two +/-10Volt analog signals. For stepper motor operation, the controller generates a step and direction signal.

Communication

The communication interface with the DMC-3425 consists of one RS-232 port (19.2 kbaud) and one 10base-T Ethernet port.

General I/O

The DMC-3415 provides interface circuitry for 7 TTL inputs and 3 TTL outputs. In addition, the controller provides two 12-bit analog inputs. The general inputs can also be used for triggering a high-speed positional latch for each axis.

NOTE: In order to accommodate 2 axes on the DMC-3425, many of the general I/O features become dedicated I/O for the second axis. The standard DMC-3425 will have 3 TTL inputs, 3 TTL outputs and 2 analog inputs. If extra I/O is needed, the DB-14064 I/O daughter card increases general purpose I/O by 64 points.

System Elements

As shown in Fig. 1.2, the DMC-3425 is part of a motion control system, which includes amplifiers, motors and encoders. These elements are described below.

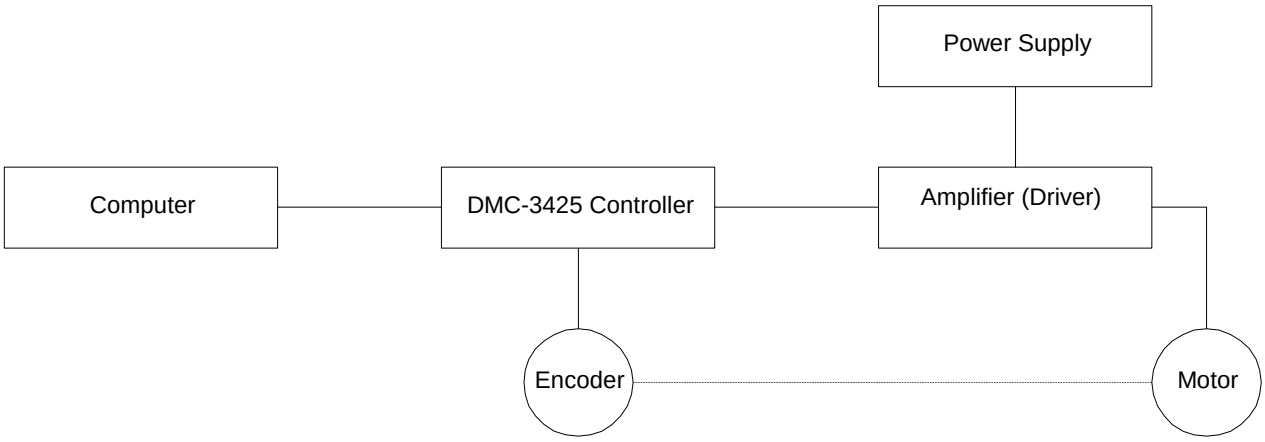


Figure 1.2 - Elements of Servo systems

Motor

A motor converts current into torque, which produces motion. Each axis of motion requires a motor sized properly to move the load at the required speed and acceleration. (Galil's "Motion Component Selector" software can help you with motor sizing). Contact Galil for more information.

The motor may be a step or servo motor and can be brush-type or brushless, rotary or linear. For step motors, the controller is capable of controlling full-step, half-step, or microstep drives. An encoder is not required when step motors are used.

Amplifier (Driver)

For each axis, the power amplifier converts a +/-10 Volt signal from the controller into current to drive the motor. For stepper motors, the amplifier converts step and direction signals into current. The amplifier should be sized properly to meet the power requirements of the motor. For brushless motors, an amplifier that provides electronic commutation is required or the controller must be configured to provide sinusoidal commutation. The amplifiers may be either pulse-width-modulated (PWM) or linear. They may also be configured for operation with or without a tachometer. For current amplifiers, the amplifier gain should be set such that a 10 Volt command generates the maximum required current. For example, if the peak motor current is 10A, the amplifier gain should be 1 A/V. For velocity mode amplifiers, 10 Volts should run the motor at the maximum speed.

For step motors, the amplifiers should accept step and direction signals.

Encoder

An encoder translates motion into electrical pulses that are fed back into the controller. The DMC-3425 accepts feedback from either a rotary or linear encoder. Typical encoders provide two channels in quadrature, known as CHA and CHB. This type of encoder is known as a quadrature encoder. Quadrature encoders may be either single-ended (CHA and CHB) or differential (CHA,CHA-, CHB,CHB-). The DMC-3425 decodes either type into quadrature states or four times the number of cycles. Encoders may also have a third channel (or index) for synchronization. The DMC-3425 can also interface to encoders with pulse and direction signals.

There is no limit on encoder line density; however, the input frequency to the controller must not exceed 3,000,000 full encoder cycles/second (12,000,000 quadrature counts/sec). For example, if the encoder line density is 10000 cycles per inch, the maximum speed is 300 inches/second. If higher encoder frequency is required, please consult the factory.

The standard voltage level is TTL (zero to five volts), however, voltage levels up to 12 Volts are acceptable. (If using differential signals, 12 Volts can be input directly to the DMC-3425. Single-ended 12 Volt signals require a bias voltage input to the complementary inputs.)

The DMC-3425 can accept analog feedback instead of an encoder for any axis. For more information see description of analog feedback in the Command Reference under the AF command.

To interface with other types of position sensors such as resolvers or absolute encoders, Galil can customize the controller and command set. Please contact Galil to talk to one of our applications engineers about your particular system requirements.

Watch Dog Timer

The DMC-3425 provides an internal watch dog timer which checks for proper microprocessor operation. The timer toggles the Amplifier Enable Output (AEN), which can be used to switch the amplifiers off in the event of a serious DMC-3425 failure. The AEN output is normally high. During power-up and if the microprocessor ceases to function properly, the AEN output will go low. The error light for each axis will also turn on at this stage. A reset is required to restore the DMC-3425 to normal operation. Consult the factory for a Return Materials Authorization (RMA) number if your DMC-3425 is damaged.

Chapter 2 Getting Started

The DMC-3425 Motion Controller

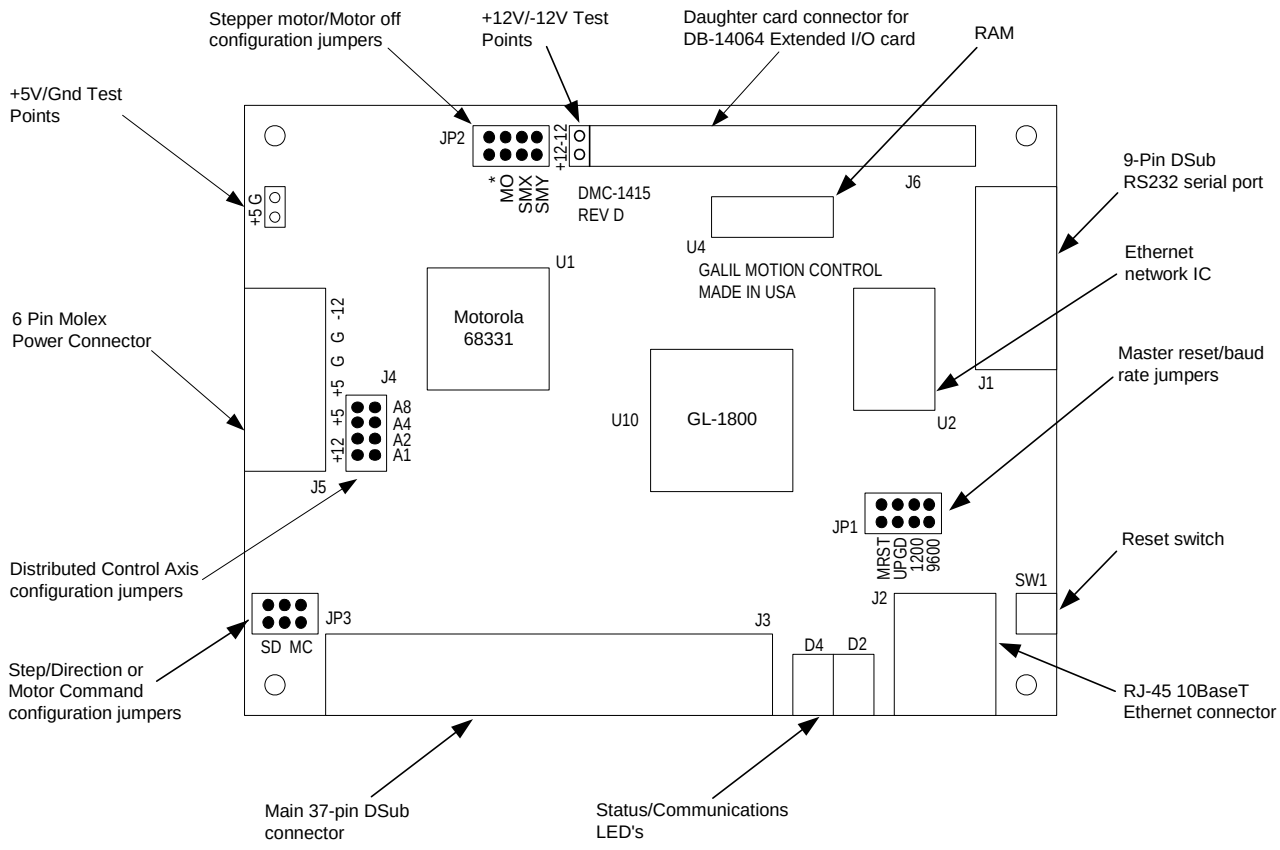


Figure 2.1 – Outline of the DMC-3425

Elements You Need

Before you start, you must get all the necessary system elements. These include:

1. (1) DMC-3425 or DMC-3415, (1) 37-pin cable (order Cable -37).
2. Servo motor(s) with encoders or stepper motors.
3. Appropriate motor drive - servo amp (Power Amplifier or AMP-1460) or stepper drive.
4. Power Supply for Amplifier
5. +5V, ±12V supply for DMC-3425
6. Communication CD from Galil
7. WSDK Servo Design Software (not necessary, but strongly recommended)
8. Interface Module ICM-1460 with screw-type terminals or integrated Interface Module/Amplifier, AMP-1460. (Note: An interconnect module is not necessary, but strongly recommended.) Also, the AMP-1460 only provides for 1 axis power amplification.

The motors may be servo (brush or brushless type) or steppers. The driver (amplifier) should be suitable for the motor and may be linear or pulse-width-modulated and it may have current feedback or voltage feedback.

For servo motors, the drivers should accept an analog signal in the +/-10 Volt range as a command. The amplifier gain should be set so that a +10V command will generate the maximum required current. For example, if the motor peak current is 10A, the amplifier gain should be 1 A/V. For velocity mode amplifiers, a command signal of 10 Volts should run the motor at the maximum required speed.

For step motors, the driver should accept step and direction signals. For start-up of a step motor system refer to Step 8c "Connecting Step Motors".

The WSDK software is highly recommended for first time users of the DMC-3425. It provides step-by-step instructions for system connection, tuning and analysis.

Installing the DMC-3425 Controller

Installation of a complete, operational DMC-3425 system consists of 9 steps.

Step 1. Determine overall motor configuration.

Step 2. Configuring jumpers on the DMC-3425.

Step 3. Connect the DC power supply and serial cable to the DMC-3425.

Step 4. Install the communications software.

Step 5. Establish communications between the DMC-3425 and the host PC.

Step 6. Set-up axis for sinusoidal commutation.

Step 7. Make connections to amplifier and encoder.

Step 8a. Connect standard servo motor.

Step 8b. Connect brushless motor for sinusoidal commutation.

Step 8c. Connect step motor.

Step 9. Tune servo system.

Step 10. Configure distributed control system.

Step 1. Determine Overall Motor Configuration

Before setting up the motion control system, the user must determine the desired motor configuration. The DMC-3425 can control standard brush or brushless servo motors, sinusoidally commutated brushless motors or stepper motors. For control of other types of actuators, such as hydraulics, please contact Galil. The following configuration information is necessary to determine the proper motor configuration:

Standard Servo Motor Operation:

The DMC-3425 has been setup by the factory for standard servo motor operation providing an analog command signal of +/- 10 volt. The position of the jumpers at JP2/JP3 determines the type of output the controllers will provide, analog motor command or PWM output. The installation of these jumpers is discussed in the section "Configuring Jumpers on the DMC-3425". Figure 2.2 shows how the jumpers are configured for the standard output mode.

The DMC-3425 controller will output the analog command signal to either brush or brushless servo amplifiers. Please note that if the brushless amplifier provides the sinusoidal commutation, the standard servo motor operation from the controller will be used. If the commutation is to be performed by the controller, please see below.

Sinusoidal Commutation:

Please consult the factory before operating with sinusoidal commutation.

Sinusoidal commutation is configured through a single software command, BA. This setting causes the controller to reconfigure the control axis to output two commutated phases. The DMC-3425 requires two DAC outputs for a single axis of commutation. Issuing the BA command will enable the second DAC for commutation.

If a DMC-3425 is used for sinusoidal commutation, the second axis will be used for the second DAC phase. Please note that if the DMC-3425 is used for sinusoidal commutation, it will still be represented by two axes within the distributed system, even though only one axis is truly active. The DMC-3415 in brushless mode will take only a single axis within the distributed system.

Further instruction for sinusoidal commutation connections are discussed in Step 6.

Stepper Motor Operation:

The DMC-3415 can be configured to operate in stepper mode by installing a hardware jumper and issuing a software command. The DMC-3425 can be configured to operate with two stepper motors by ordering the DMC-3425-Stepper option from the factory. To configure the DMC-3425 for stepper motor operation, the controller requires a jumper for the stepper motors and the command, MT, must be given. The installation of the stepper motor jumper is discussed in the following section entitled "Configuring Jumpers on the DMC-3425". Further instructions for stepper motor connections are discussed in Step 8b.

Step 2. Configuring Jumpers on the DMC-3425

Master Reset and Upgrade Jumper

JP1 contains two jumpers, MRST and UPGD. The MRST jumper is the Master Reset jumper. When MRST is connected, the controller will perform a master reset upon PC power up or upon the reset input going low. Whenever the controller has a master reset, all programs, arrays, variables, and motion control parameters stored in EEPROM will be ERASED.

The UPGD jumper enables the user to unconditionally update the controller's firmware. This jumper is not necessary for firmware updates when the controller is operating normally, but may be necessary in cases of corrupted EEPROM. EEPROM corruption should never occur, however, it is possible if

there is a power fault during a firmware update. If EEPROM corruption occurs, your controller may not operate properly. In this case, install the UPGD Jumper and use the update firmware function on the Galil Smart Terminal or WSDK to re-load the system firmware.

Setting the Baud Rate on the DMC-3425

The jumpers labeled "9600" and "1200" at JP1 allow the user to select the serial communication baud rate. The baud rate can be set using the following table:

JUMPER SETTINGS		BAUD RATE
9600	**1200**	**--**
OFF	OFF	19200
ON	OFF	9600
OFF	ON	1200

The default baud rate for the controller is 19.2k.

Selecting MO as default on the DMC-3425

The default condition for the motor on the DMC-3425 is the servo on (SH) state. This will enable the amplifiers upon power up of the controller. This state can be changed to the motor off (MO) default by placing a jumper at JP2 across the MO terminals. This will power up the controller with the amplifiers disabled and the motor command off. The SH command must then be given in order for the servos or steppers to operate.

Stepper Motor Jumpers

The DMC-3415 is user configurable to control either a servo motor or a stepper motor. The DMC-3425 is factory default to servo control, but may also control two steppers if ordered from the factory as a DMC-3425-Stepper.

To configure the DMC-3415 for stepper output, two jumpers must be placed on the controller. First, the SMX jumper at location JP2 must be installed. This configures the board for step/direction output. Second, the jumpers at location JP3 must be moved from the MC position to the SD position as shown in Figure 2.2. This configures the output pins on the controller to output step and direction instead of the analog motor command.

The configuration for two stepper motors on the DMC-3425-Stepper is handled at the factory. The same procedure is used, placing jumpers on SMX and SMY at location JP2, and moving the SD/MC jumpers at location JP3. A board modification is also required, which should only be handled by Galil technicians.

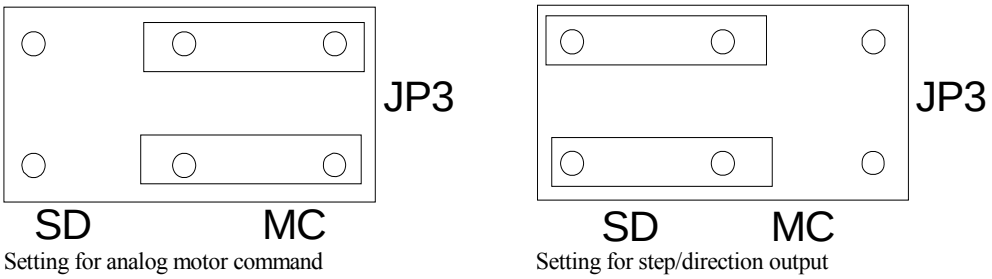


Figure 2.2 - Jumper settings for motor command output

Axis Configuration Jumpers

When using the HC automatic configuration, jumpers must be set to indicate which controller is the master and which controllers are slaves. Depending on the configuration of the jumpers, a controller will be set up as either the A (B) master or any of the axes slaves.

The 8-pin jumper, found at location J4 next to the Molex power connector, is used to select axes configurations. Jumpers at this location are labeled A1, A2, A4 and A8, which represent the binary value for each of the 8 axes within a system. The following chart shows proper jumper selection for each of the DMC-3415 or DMC-3425's in a system.

Master A (B) axis	No Jumpers			
Slave Axis B	A1 On	A2 Off	A4 Off	A8 Off
Slave Axes C	A1 Off	A2 On	A4 Off	A8 Off
Slave Axis D	A1 On	A2 On	A4 Off	A8 Off
Slave Axes E	A1 Off	A2 Off	A4 On	A8 Off
Slave Axis F	A1 On	A2 Off	A4 On	A8 Off
Slave Axes G	A1 Off	A2 On	A4 On	A8 Off
Slave Axis H	A1 On	A2 On	A4 On	A8 Off

Jumpers on a card are used to denote the first axis it represents in a system. Therefore, a DMC-3415 takes up a single jumper setting. A DMC-3425 is selected with a single jumper setting but represents two axes.

For example, the jumper settings for a system with a DMC-3415 master A axis, a DMC-3425 slave BC axis and a DMC-3415 slave D axis, the following jumper settings would be used.

Master A – No Jumpers			
Slave Axis BC – A1 On	A2 Off	A4 Off	A8 Off
Slave Axis D – A1 On	A2 On	A4 Off	A8 Off

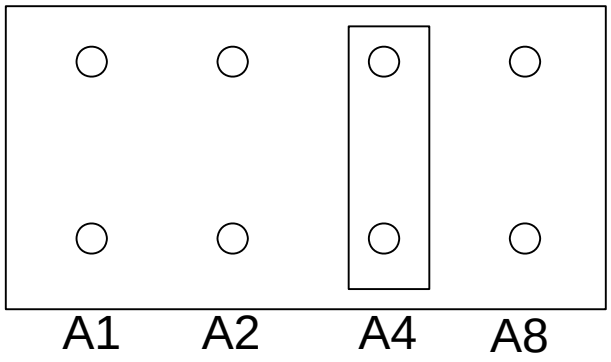


Fig. 2.3 – Example jumper settings for DMC-3425 E, F axis configuration.

Step 3. Connecting AC or DC power and the Serial Cable to the DMC-3425

1. Insert 37-pin cable to J3. Connect the other end of the cable to the ICM-1460.
2. If using serial communications, use the 9-pin RS232 ribbon cable to connect the SERIAL port of the DMC-3425 to your computer or terminal communications port. The DMC-3425 serial

port is configured as DATASET. Your computer or terminal must be configured as a DATATERM for full duplex, no parity, 8 bits data, one start bit and one stop bit.

Your computer needs to be configured as a "dumb" terminal that sends ASCII characters as they are typed to the DMC-3425.

Connections to the controller for Ethernet communication are covered in Step 5.

3. If using the card level version, apply ±12V and +5V power to the J5 connector. If using the box level version, connect the AC cord to a power outlet. AC power requirements for the controller are single phase, 50 or 60 Hz at 90 to 260 VAC.
4. Applying power will turn on the green LED power indicator.

Step 4. Installing the Communications Software

After applying power to the computer, you should install the Galil software that enables communication between the controller and PC.

Using DOS:

Using the Galil Software CD-ROM, go to the directory, DMCDOS. Type "INSTALL" at the DOS prompt and follow the directions.

Using Windows 3.x (16 bit versions):

Using the Galil Software CD ROM, go to the directory, DMCWIN16. Run DMCWIN16.exe at the Command prompt and follow the directions.

Using Windows 95, NT or 98 (32 bit versions):

The Galil Software CD-ROM will automatically begin the installation procedure when the CD-ROM is installed. After installing the Galil CD-ROM software on your computer, you can easily install other software components as desired. To install the basic communications software, run the Galil Software CD-ROM and choose “DMC Smart Terminal”. This will install the Galil Terminal that can be used for communication.

Step 5. Establishing Communication between the DMC-3425 and the host PC

Note: This section will show how to communicate with a single DMC-3425 or DMC-3415 controller. If the controllers will be configured in a multi-axis, distributed control system, only the master axis needs an IP address actively configured.

Communicating through the RS-232 Serial Communications Port

Connect the DMC-3425 serial port to your computer via the Galil CABLE-9PIN-D (RS-232 Cable).

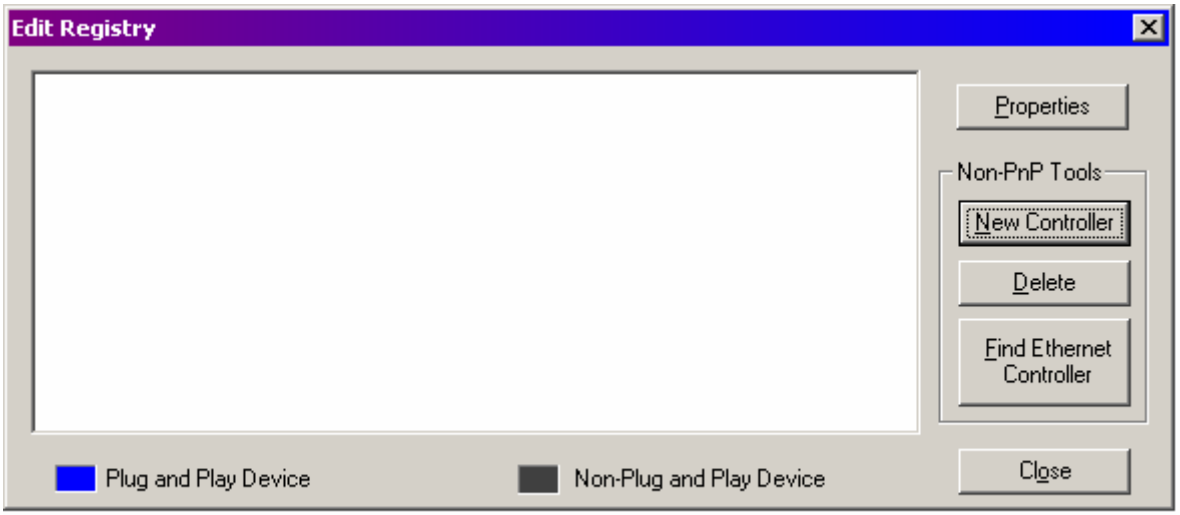
Using Galil Software for DOS

To communicate with the DMC-3425, type TALK2DMC at the prompt. Once you have established communication, the terminal display should show a colon, :. If you do not receive a colon, press the carriage return. If a colon prompt is not returned, there is most likely an incorrect setting of the serial communications port. The user must ensure that the correct communication port and baud rate are specified when attempting to communicate with the controller. Please note that the serial port on the controller must be set for handshake mode for proper communication with Galil software. The user must also insure that the proper serial cable is being used. See appendix for pin-out of serial port.

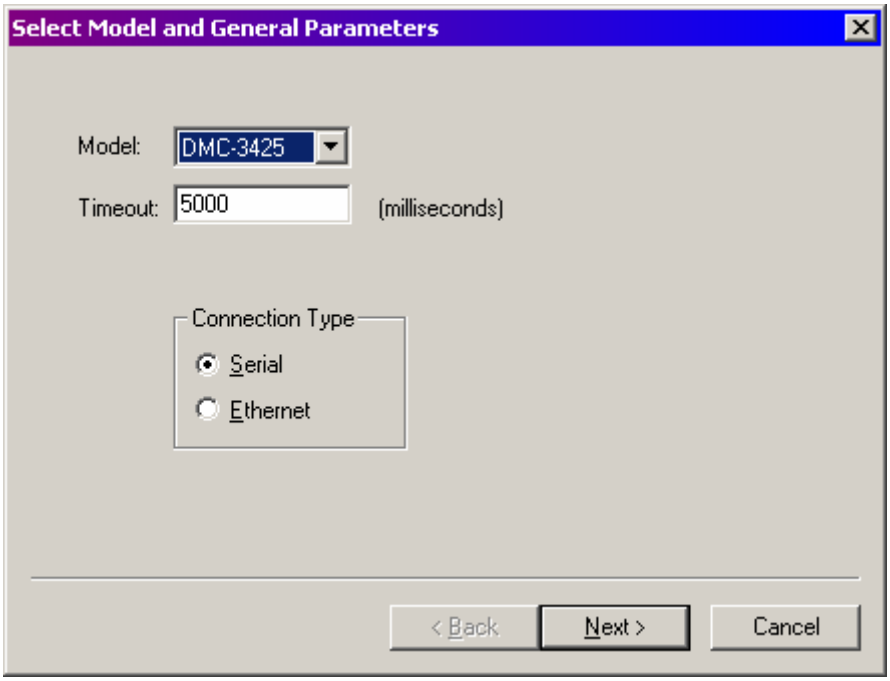
Using Galil Software for Windows

In order for the Windows software to communicate with a Galil controller, the controller must be registered in the Windows Registry. To register a controller, you must specify the model of the controller, the communication parameters, and other information. The registry is accessed through the Galil software, such as WSDK or DMCSmartTerm.

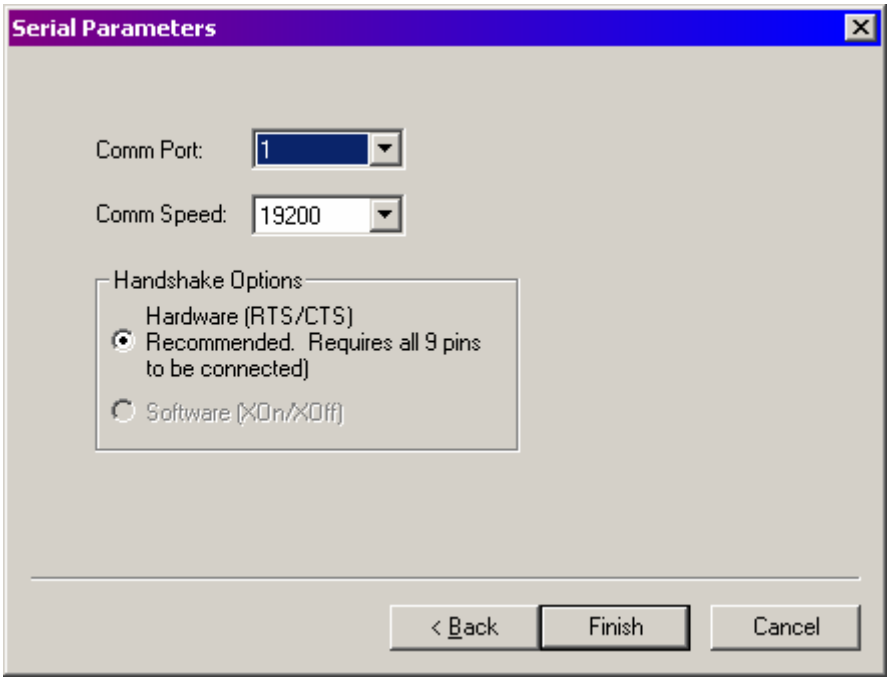
The registry window is equipped with a button to Add a New Controller, change the Properties of an existing controller, Delete a controller, or Find an Ethernet controller.



Use the **New Controller** button to add a new entry to the Registry. Use the **Properties** button to change the properties of a current controller. For a new registration, you will need to supply the Galil Controller type. The controller model number must be entered. If you are changing an existing controller, this field will already have an entry. Pressing the down arrow to the right of this field will reveal a menu of valid controller types. Once the DMC-3425 has been selected, there is a choice for either **Serial** or **Ethernet** communication, as shown below. Select **Serial** communication.



After selecting **Next**, the registry information will show a default Comm Port of 1 and a default Comm Speed of 19200 appears. This information should be changed as necessary to reflect the computers Comm Port and the baud rate set by the controller's baud rate jumpers.



Once you have set the appropriate Registry information for your controller, Select **Finish** and close the registry window. You will now be able to communicate with the DMC-3425. Within WSDK, select **File** and **Connect to Controller**. Within DMCSmartTerm, select **Tools** and **Select Controller**. Once

the entry has been selected, click on the **OK** button. If the software has successfully established communications with the controller, the registry entry will be displayed at the top of the screen.

If you are not properly communicating with the controller, the program will pause for 3-15 seconds. The top of the screen will display the message "Status: not connected with Galil motion controller" and the following error will appear: "STOP - Unable to establish communication with the Galil controller. A time-out occurred while waiting for a response from the Galil controller." If this message appears, you must click OK. In this case, there is most likely an incorrect setting of the serial communications port. The user must ensure that the correct communication port and baud rate are specified when attempting to communicate with the controller. Please note that the serial port on the controller must be set for handshake mode for proper communication with Galil software. The user must also insure that the proper straight-through serial cable is being used (no Null modem). See appendix for the correct pin-outs for the serial cable.

Once you establish communications, click on the menu for terminal and you will receive a colon prompt. Communicating with the controller is described in later sections.

Using Non-Galil Communication Software

The DMC-3425 serial port is configured as DATASET. Your computer or terminal must be configured as a DATATERM for full duplex, no parity, 8 data bits, one start bit and one stop bit.

Check to insure that the baud rate switches have been set to the desired baud rate as described above.

Your computer needs to be configured as a "dumb" terminal that sends ASCII characters as they are typed to the DMC-3425. Use the EO command to specify if the characters should be echoed back from the controller.

Sending Test Commands to the Terminal:

After you connect your terminal, press <carriage return> or the <enter> key on your keyboard. In response to carriage return (CR), the controller responds with a colon, :

Now type

TPA (CR)

This command directs the controller to return the current position of the A axis. The controller should respond with a number such as

0000000

Communicating through the Ethernet

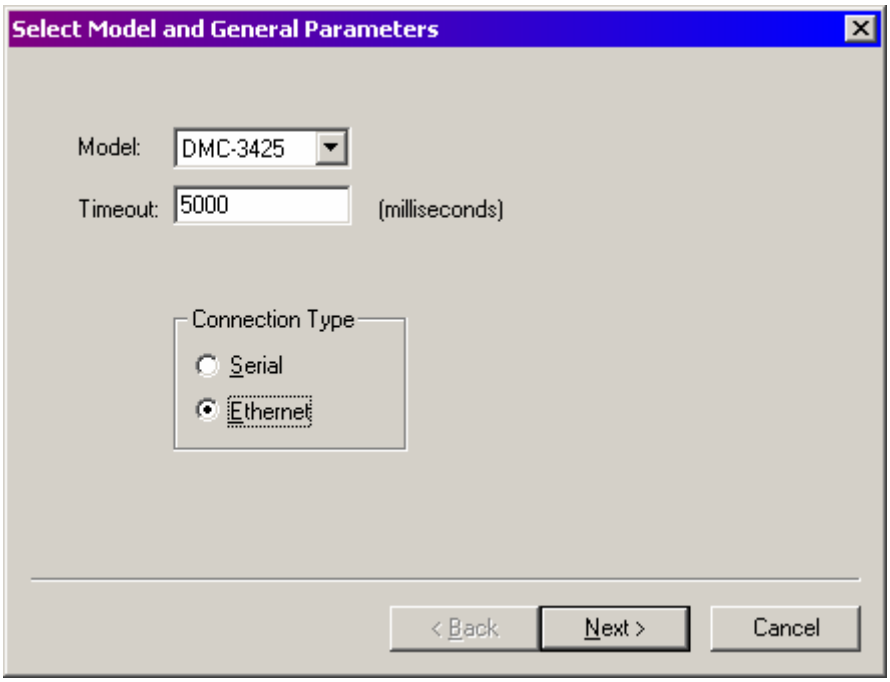
For Ethernet communication, connect the DMC-3425 to your computer or to a hub. If connecting through a switch or a hub, a standard RJ45 Ethernet cable is used. If connecting directly to the PC, a cross-over RJ45 Ethernet cable must be used.

Using Galil Software for Windows

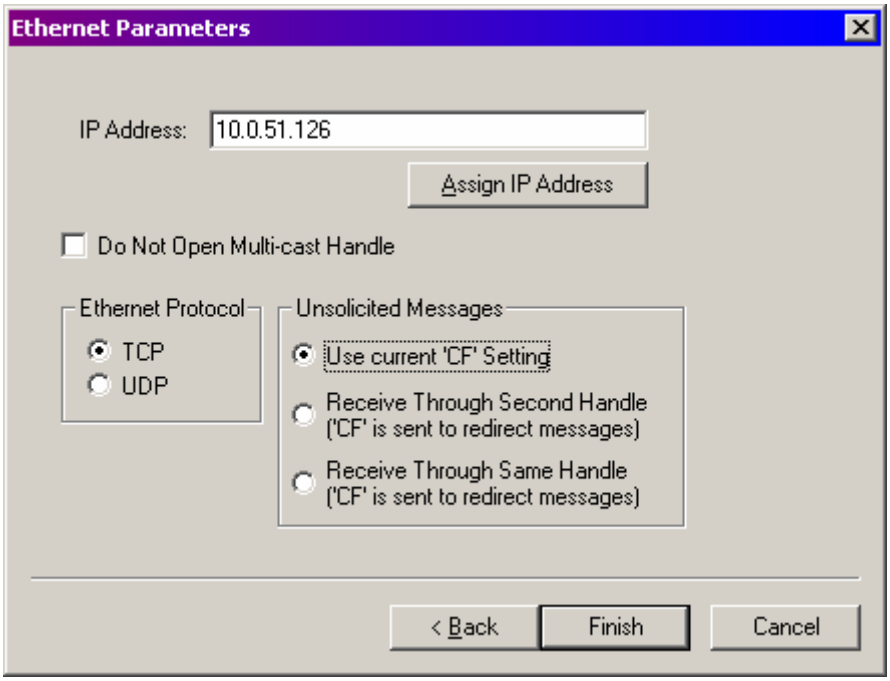
The controller must be registered in the Galil Windows registry for the host computer to communicate with it. The registry may be accessed via Galil software, such as WSDK or DMCSmartTerm.

From WSDK, the registry is accessed under the FILE menu. From DMCSmartTerm it is accessed under the **Tools** and **Controller Registration** menu. In the Galil Registry, the DMC-3425 can either be added manually with the **New Controller** button or the software can automatically try to find the controller with the **Find Ethernet Controller** button.

The first registry option is to use the **New Controller** button. The DMC-3425 should be selected from the models listed, with Ethernet selected as the mode of communication.



After **Next** is pressed, the next screen will allow the IP address to be selected and assigned.



Enter the IP address obtained from your system administrator into the box **IP Address**. Select the button corresponding to the protocol in which you wish to communicate with the controller, UDP or TCP. If the IP address has not been already assigned to the controller, click on **ASSIGN IP ADDRESS.**

ASSIGN IP ADDRESS will check the controllers that are linked to the network to see which ones do not have an IP address. The program will then ask you whether you would like to assign the IP address you entered to the controller with the specified serial number. Click on **YES** to assign it, **NO** to move to next controller, or **CANCEL** to not save the changes. If there are no controllers on the network that do not have an IP address assigned, the program will state this. Once the correct controller has been selected, click on **Finish.**

If an IP address has already been assigned to the controller through the serial port and the IA command, add this address to the **IP Address** box and then select **Finish.**

The second method for registering the controller is by using the option within the registry labeled **Find Ethernet Controllers.** This utility uses the DMCNet software program to search for any controllers on the network, both with and without IP addresses. If the DMC-3425 does not have an IP address, the utility will listen for the BOOTP packet and then ask for an IP address to be assigned. Once the IP address is added, click on **Register** and the controller will be added to the Galil Registry. If an IP address has already been assigned to the controller, the utility will list that controller with its current IP address. At this point, click on **Register** and the controller will be added to the Galil Registry.

Once you have set the appropriate Registry information for your controller, Select **Close** to close the registry window. You will now be able to communicate with the DMC-3425. Within WSDK, select **File** and **Connect to Controller**. Within DMCSmartTerm, select **Tools** and **Select Controller**. Once the appropriate entry has been selected, click on the **OK** button. If the software has successfully established communications with the controller, the registry entry will be displayed at the top of the screen.

See Chapter 4 Communication for additional information on the Ethernet configuration and connection.

Sending Test Commands to the Terminal:

After you connect your terminal, press <return> or the <enter> key on your keyboard. In response to carriage return <return>, the controller responds with a colon, :

Now type

TPA <return>

This command directs the controller to return the current position of the A axis. The controller should respond with a number such as

0000000

Step 6. Set-up axis for sinusoidal commutation (optional)

* This step is only required when the controller will be used to control a brushless motor with sinusoidal commutation. **Please consult the factory before operating with sinusoidal commutation.**

The command BA is used to specify sinusoidal commutation mode for the DMC-3415 or DMC-3425. In this mode the controller will output two sinusoidal phases for the DACs. Once specified, follow the procedure outlined in Step 8b.

Step 7. Make connections to amplifier and encoder

Once you have established communications between the software and the DMC-3425, you are ready to connect the rest of the motion control system. The motion control system generally consists of an ICM-1460 Interface Module, a servo amplifier, and a motor to transform the current from the servo amplifier into torque for motion. Galil also offers the AMP-1460 Interface Module which is an ICM-1460 equipped with a servo amplifier for a DC motor.

A signal breakout board of some type is strongly recommended. If you are using a breakout board from a third party, consult the documentation for that board to insure proper system connection.

If you are using the ICM-1460 or AMP-1460 with the DMC-3425, connect the 37-pin cable between the controller and interconnect module.

Here are the first steps for connecting a motion control system:

Step A. Connect the motor to the amplifier *with no connection to the controller*. Consult the amplifier documentation for instructions regarding proper connections. Connect and turn on the amplifier power supply. If the amplifiers are operating properly, the motor should stand still even when the amplifiers are powered up.

Step B. Connect the amplifier enable signal. Before making any connections from the amplifier to the controller, you need to verify that the ground level of the amplifier is either floating or at the same potential as earth.

Note: If you are using a DMC-3425-Stepper, the amplifier enable signal is used for the second stepper output.

WARNING: When the amplifier ground is not isolated from the power line or when it has a different potential than that of the computer ground, serious damage may result to the computer controller and amplifier.

If you are not sure about the potential of the ground levels, connect the two ground signals (amplifier ground and earth) by a 10 kΩ resistor and measure the voltage across the resistor. Only if the voltage is zero, proceed to connect the two ground signals directly.

The amplifier enable signal is used by the controller to disable the motor. This signal is labeled AMPEN on the ICM-1460 and should be connected to the enable signal on the amplifier. Note that many amplifiers designate this signal as the INHIBIT signal. Use the command, MO, to disable the motor amplifiers - check to insure that the motor amplifiers have been disabled (often this is indicated by an LED on the amplifier).

This signal changes under the following conditions: the watchdog timer activates, the motor-off command, MO, is given, or the OE1 command (Enable Off-On-Error) is given and the position error exceeds the error limit. As shown in Figure 3.1, AEN can be used to disable the amplifier for these conditions.

The standard configuration of the AEN signal is TTL active high. In other words, the AEN signal will be high when the controller expects the amplifier to be enabled. The polarity and the amplitude can be changed if you are using the ICM-1460 interface board. To change the polarity from active high (5 volts = enable, zero volts = disable) to active low (zero volts = enable, 5 volts = disable), replace the 7407 IC with a 7406. Note that many amplifiers designate the enable input as 'inhibit'.

To change the voltage level of the AEN signal, note the state of the JP1 jumper on the ICM-1460. When the jumper is placed across 5V and AEN, the output voltage is 0-5V. To change to 12 volts, pull the jumper and rotate it so that +12V is connected to AEN. If you remove the jumper, the output signal is an open collector, allowing the user to connect an external supply with voltages up to 24V.

Step C. Connect the encoders

For stepper motor operation, an encoder is optional.

For servo motor operation, if you have a preferred definition of the forward and reverse directions, make sure that the encoder wiring is consistent with that definition.

The DMC-3425 accepts single-ended or differential encoder feedback with or without an index pulse. If you are not using the AMP-1460 or the ICM-1460, you will need to consult the appendix for the encoder pinouts for connection to the motion controller. The AMP-1460 and the ICM-1460 can accept encoder feedback from a 10-pin ribbon cable or individual signal leads. For a 10-pin ribbon cable encoder, connect the cable to the protected header connector labeled JP2. For individual wires, simply match the leads from the encoder you are using to the encoder feedback inputs on the interconnect board. The signal leads are labeled CHA, CHB, and INDEX. These labels represent channel A, channel B, and the INDEX pulse, respectively. For differential encoders, the complement signals are labeled CHA-, CHB-, and INDEX-.

Note: When using pulse and direction encoders, the pulse signal is connected to CHA and the direction signal is connected to CHB. The controller must be configured for pulse and direction with the command CE. See the command summary for further information on the command CE.

Step D. Verify proper encoder operation.

Once the encoder is connected as described above, turn the motor shaft and interrogate the position with the instruction TP <return>. The controller response will vary as the motor is turned.

At this point, if TP does not vary with encoder rotation, there are three possibilities:

1. The encoder connections are incorrect - check the wiring as necessary.
2. The encoder has failed - using an oscilloscope, observe the encoder signals. Verify that both channels A and B have a peak magnitude between 5 and 12 volts. Note that if only one encoder channel fails, the position reporting varies by one count only. If the encoder failed, replace the encoder. If you cannot observe the encoder signals, try a different encoder.
3. There is a hardware failure in the controller - connect the same encoder to a different axis. If the problem disappears, you probably have a hardware failure. Consult the factory for help.

Step E. Connect Hall Sensors if available (sinusoidal commutation only)

Please consult factory before operating with sinusoidal commutation. Hall sensors are only used with sinusoidal commutation on the DMC-3415 or DMC-3425 and are not necessary for proper operation. The use of hall sensors allows the controller to automatically estimate the commutation phase upon reset and also provides the controller the ability to set a more precise commutation phase. Without hall sensors, the commutation phase must be determined manually.

The Hall effect sensors are connected to the digital inputs of the controller. These inputs can be used with the general-purpose inputs (bits 1 - 7). If you are using the DMC-3425, only the first 3 inputs are available for general purpose.

Each set of inputs must use inputs that are in consecutive order. The input lines are specified with the command, BI. For example, if the Hall sensors are connected to inputs 1, 2 and 3, use the instruction:

BI1 <CR>

Step 8a. Connect Standard Servo Motor

The following discussion applies to connecting the DMC-3425 controller to standard servo motor amplifiers:

The motor and the amplifier may be configured in the torque or the velocity mode. In the torque mode, the amplifier gain should be such that a 10 Volt signal generates the maximum required current. In the velocity mode, a command signal of 10 Volts should run the motor at the maximum required speed.

Step by step directions on servo system setup are also included on the WSDK (Windows Servo Design Kit) software offered by Galil. See section on WSDK for more details.

Check the Polarity of the Feedback Loop

It is assumed that the motor and amplifier are connected together and that the encoder is operating correctly (Step D). Before connecting the motor amplifiers to the controller, read the following discussion on setting Error Limits and Torque Limits.

Step A. Set the Error Limit as a Safety Precaution

Usually, there is uncertainty about the correct polarity of the feedback. The wrong polarity causes the motor to run away from the starting position. Using a terminal program, such as DMCSmartTerm, the following parameters can be given to avoid system damage:

Input the commands:

ER 2000,2000 <CR> Sets error limit to be 2000 counts

OE 1,1 <CR> Disables amplifier when excess error exists

If the motor runs away and creates a position error of 2000 counts, the motor amplifier will be disabled.

Note: This function requires the AEN signal to be connected from the controller to the amplifier.

Step B. Setting Torque Limit as a Safety Precaution

To limit the maximum voltage signal to your amplifier, the DMC-3425 controller has a torque limit command, TL. This command sets the maximum voltage output of the controller and can be used to avoid excessive torque or speed when initially setting up a servo system.

When operating an amplifier in torque mode, the voltage output of the controller will be directly related to the torque output of the motor. The user is responsible for determining this relationship using the documentation of the motor and amplifier. The torque limit can be set to a value that will limit the motors output torque.

When operating an amplifier in velocity or voltage mode, the voltage output of the controller will be directly related to the velocity of the motor. The user is responsible for determining this relationship using the documentation of the motor and amplifier. The torque limit can be set to a value that will limit the speed of the motor.

For example, the following command will limit the output of the controller to 1 volt:

TL 1 <CR> Sets torque limit to 1 Volt on A axis

Note: Once the correct polarity of the feedback loop has been determined, the torque limit should, in general, be increased to the default value of 9.99. The servo will not operate properly if the torque limit is below the normal operating range. See description of TL in the command reference.

Step C. Disable motor

Issue the motor off command to disable the motor.

MO <CR> Turns motor off

Step D. Connecting the Motor

Once the parameters have been set, connect the analog motor command signal (ACMD) to the amplifier input.

Issue the servo here command to turn the motors on. To test the polarity of the feedback, command a move with the instruction:

SH <CR>	Servo Here to turn motors on
PR 1000 <CR>	Position relative 1000 counts
BG <CR>	Begin motion

When the polarity of the feedback is wrong, the motor will attempt to run away. The controller should disable the motor when the position error exceeds 2000 counts. In this case, the polarity of the loop must be inverted.

Inverting the Loop Polarity

When the polarity of the feedback is incorrect, the user must invert the loop polarity and this may be accomplished by several methods. If you are driving a brush-type DC motor, the simplest way is to invert the two motor wires (typically red and black). For example, switch the M1 and M2 connections going from your amplifier to the motor. When driving a brushless motor, the polarity reversal may be done with the encoder. If you are using a single-ended encoder, interchange the signal CHA and CHB. If, on the other hand, you are using a differential encoder, interchange only CHA+ and CHA-. The loop polarity and encoder polarity can also be affected through software with the MT, and CE commands. For more details on the MT command or the CE command, see the Command Reference section.

Sometimes the feedback polarity is correct (the motor does not attempt to run away) but the direction of motion is reversed with respect to the commanded motion. If this is the case, reverse the motor leads AND the encoder signals.

If the motor moves in the required direction but stops short of the target, it is most likely due to insufficient torque output from the motor command signal ACMD. This can be alleviated by reducing system friction on the motors. The instruction:

TT <CR>	Tell torque

reports the level of the output signal. It will show a non-zero value that is below the friction level.

Once you have established that you have closed the loop with the correct polarity, you can move on to the compensation phase (servo system tuning) to adjust the PID filter parameters, KP, KD and KI. It is necessary to accurately tune your servo system to ensure fidelity of position and minimize motion oscillation as described in the next section.

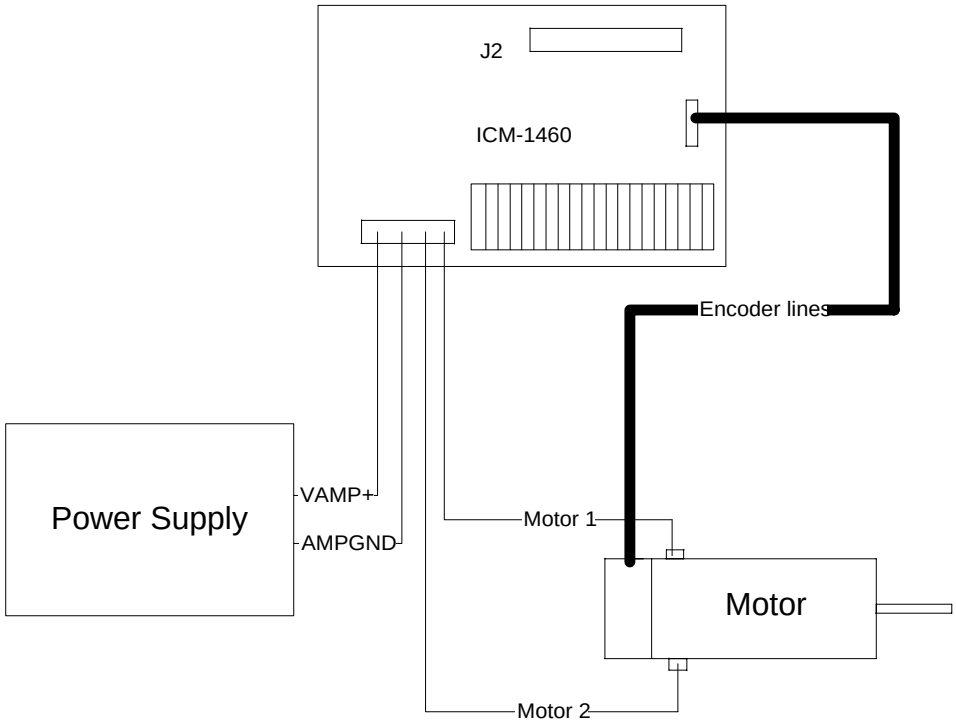


Figure 2.3 - System Connections with the AMP-1460 Amplifier

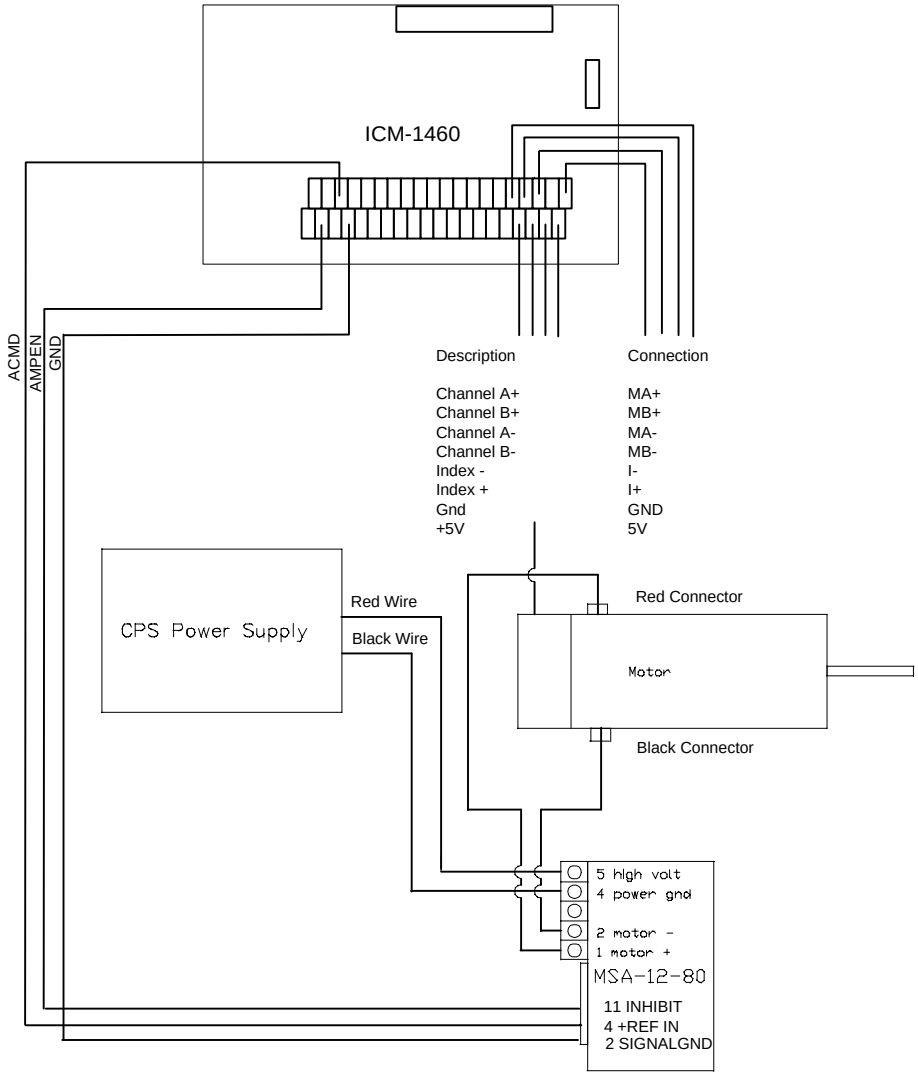


Figure 2.4 - System Connections with a separate amplifier (MSA 12-80). This diagram shows the connections for a standard DC Servo Motor and encoder.

Step 8b. Connect brushless motor for sinusoidal commutation

Please consult the factory before operating with sinusoidal commutation. Any controller within the distributed system may be configured for sinusoidal commutation. If a DMC-3415 is used, the second DAC is simply initiated with the BA command. If a DMC-3425 is used, it will control only a single brushless motor, but will take up two axes in configuration. When using sinusoidal commutation, the parameters for the commutation must be determined and saved in the controller's non-volatile memory. The servo can then be tuned as described in Step 9.

Step A. Disable the motor amplifier

Use the command, MO, to disable the motor amplifiers.

Step B. Connect the motor amplifier to the controller.

The sinusoidal commutation amplifier requires 2 signals, usually denoted as Phase A and Phase B. These inputs should be connected to the two sinusoidal signals

generated by the controller. The first signal is the main controller motor output, ACMD. The second signal utilizes the second DAC on the controller and is brought out on the ICM-1460 at pin 38 (ACMD2).

It is not necessary to be concerned with cross-wiring the 1st and 2nd signals. If this wiring is incorrect, the setup procedure will alert the user (Step D).

Step C. Specify the Size of the Magnetic Cycle.

Use the command, BM, to specify the size of the brushless motors magnetic cycle in encoder counts. For example, if you are using a linear motor where the magnetic cycle length is 62 mm, and the encoder resolution is 1 micron, the cycle equals 62,000 counts. This can be commanded with the command:

BM 62000 <CR>

On the other hand, if you are using a rotary motor with 4000 counts per revolution and 3 magnetic cycles per revolution (three pole pairs) the command is:

BM 1333.333 <CR>

Step D. Test the Polarity of the DACs and Hall Sensor Configuration.

Use the brushless motor setup command, BS, to test the polarity of the output DACs. This command applies a certain voltage, V, to each phase for some time T, and checks to see if the motion is in the correct direction.

The user must specify the value for V and T. For example, the command:

BS 2,700 <CR>

will test the brushless axis with a voltage of 2 volts, applying it for 700 milliseconds for each phase. In response, this test indicates whether the DAC wiring is correct and will indicate an approximate value of BM. If the wiring is correct, the approximate value for BM will agree with the value used in the previous step.

Note: In order to properly conduct the brushless setup, the motor must be allowed to move a minimum of one magnetic cycle in both directions.

Note: When using Galil Windows software, the timeout must be set to a minimum of 10 seconds (time-out = 10000) when executing the BS command. This allows the software to retrieve all messages returned from the controller.

If Hall Sensors are Available:

Since the Hall sensors are connected randomly, it is very likely that they are wired in the incorrect order. The brushless setup command indicates the correct wiring of the Hall sensors. The hall sensor wires should be re-configured to reflect the results of this test.

The setup command also reports the position offset of the hall transition point and the zero phase of the motor commutation. The zero transition of the Hall sensors typically occurs at 0°, 30° or 90° of the phase commutation. It is necessary to inform the controller about the offset of the Hall sensor and this is done with the instruction, BB.

Step E. Save Brushless Motor Configuration

It is very important to save the brushless motor configuration in non-volatile memory. After the motor wiring and setup parameters have been properly configured, the burn command, BN, should be given.

If Hall Sensors are Not Available:

Without hall sensors, the controller will not be able to estimate the commutation phase of the brushless motor. In this case, the controller could become unstable until the commutation phase has been set using the BZ command (see next step). It is highly recommended that the motor off command be given before executing the BN command. In this case, the motor will be disabled upon power up or reset and the commutation phase can be set before enabling the motor.

Step F. Set Zero Commutation Phase

When an axis has been defined as sinusoidally commutated, the controller must have an estimate for commutation phase. When hall sensors are used, the controller automatically estimates this value upon reset of the controller. If no hall sensors are used, the controller will not be able to make this estimate and the commutation phase must be set before enabling the motor.

If Hall Sensors are Not Available:

To initialize the commutation without Hall effect sensor use the command, BZ. This function drives the motor to a position where the commutation phase is zero, and sets the phase to zero.

The BZ command argument is a real number that represents the voltage to be applied to the amplifier during the initialization. When the voltage is specified by a positive number, the initialization process will end up in the motor off (MO) state. A negative number causes the process to end in the Servo Here (SH) state.

Warning: This command must move the motor to find the zero commutation phase. This movement is instantaneous and will cause the system to jerk. Larger applied voltages will cause more severe motor jerk. The applied voltage will typically be sufficient for proper operation of the BZ command. For systems with significant friction, this voltage may need to be increased and for systems with very small motors, this value should be decreased.

For example,

BZ -2 <CR>

will drive the axis to zero, using a 2V signal. The controller will then leave the motor enabled. For systems that have external forces working against the motor, such as gravity, the BZ argument must provide a torque 10x the external force. If the torque is not sufficient, the commutation zero may not be accurate.

If Hall Sensors are Available:

The estimated value of the commutation phase is good to within 30°. This estimate can be used to drive the motor but a more accurate estimate is needed for efficient motor operation. There are 3 possible methods for commutation phase initialization:

Method 1. Use the BZ command as described above.

Method 2. Drive the motor close to commutation phase of zero and then use BZ command. This method decreases the amount of system jerk by moving the motor close to zero commutation phase before executing the BZ command. The controller makes an estimate for the number of encoder counts between the current position and the position of zero commutation phase. This value is stored in the operand _BZx. Using this operand the controller can be commanded to move the motor. The BZ command is then issued as described above. For example, to initialize the A axis motor upon power or reset, the following commands may be given:

SH <CR> Enable A axis motor

PRA=-1*(_BZA) <CR>	Move A motor close to zero commutation phase
BGA <CR>	Begin motion on A axis
AMA<CR>	Wait for motion to complete on A axis
BZA=-1 <CR>	Drive motor to commutation phase zero and leave motor on

Method 3. Use the command, BC. This command uses the hall transitions to determine the commutation phase. Ideally, the hall sensor transitions will be separated by exactly 60° and any deviation from 60° will affect the accuracy of this method. If the hall sensors are accurate, this method is recommended. The BC command monitors the hall sensors during a move and monitors the Hall sensors for a transition point. When that occurs, the controller computes the commutation phase and sets it. For example, to initialize the motor upon power or reset, the following commands may be given:

SH <CR>	Enable motor
BC <CR>	Enable the brushless calibration command
PR 50000 <CR>	Command a relative position movement
BG <CR>	Begin motion. When the hall sensors detect a phase transition, the commutation phase is re-set.

Step 8c. Connect Step Motors

In Stepper Motor operation, the pulse output signal has a 50% duty cycle. Step motors operate open loop and do not require encoder feedback. When a stepper is used, the auxiliary encoder for the corresponding axis is unavailable for an external connection. If an encoder is used for position feedback, connect the encoder to the main encoder input corresponding to that axis. The commanded position of the stepper can be interrogated with RP or DE. The encoder position can be interrogated with TP. Only the DMC-3415 allows the use of the main encoder input with a stepper motor. The DMC-3425 does not have this option.

The frequency of the step motor pulses can be smoothed with the filter parameter, KS. The KS parameter has a range between 0.5 and 8, where 8 implies the largest amount of smoothing. *See Command Reference regarding KS*.

The DMC-3425 profiler commands the step motor amplifier. All DMC-3425 motion commands apply such as PR, PA, VP, CR and JG. The acceleration, deceleration, slew speed and smoothing are also used. Since step motors run open-loop, the PID filter does not function and the position error is not generated.

To connect step motors with the DMC-3425 you must follow this procedure:

Step A. Install SM jumper

Install the jumper SMX at location JP2 to enable stepper motor operation on the DMC-3415. For the DMC-3425-Stepper, the jumpers should be loaded on SMX and SMY. For a discussion of SM jumpers, see section “Step 2. Configuring Jumpers on the DMC-3425”.

Step B. Connect step and direction signals from the controller to respective signals on your step motor amplifier.

The DMC-3415 outputs STEPX (step) signals on the ICM-1460 terminal labeled ACMD, and outputs DIRX (direction) signals on the ICM-1460 terminal labeled ACMD2.

The DMC-3425 outputs STEPY signals on the ICM-1460 terminal labeled ERROR, and outputs DIRX on the ICM-1460 terminal labeled AMPEN. X-axis connections are identical to the DMC-3415.

Consult the documentation for your step motor amplifier for proper connections.

Step C. Configure DMC-3425 for motor type using MT command. You can configure the DMC-3425 for active high or active low pulses. Use the command MT 2 for active high step motor pulses and MT -2 for active low step motor pulses. *See description of the MT command in the Command Reference.*

Note: The DMC-3425 must be ordered as a DMC-3425-Stepper to drive two axes of stepper motors.

Step 9. Tune the Servo System

The system compensation provides fast and accurate response by adjusting the filter parameters. The following presentation suggests a simple and easy way for compensation. More advanced design methods are available with software design tools from Galil, such as the Windows Servo Design Kit (WSDK software).

If the torque limit was set as a safety precaution in the previous step, you may want to increase this value. See Step B of the above section "*Setting Torque Limit as a Safety Precaution*"

The filter has three parameters: the damping, KD; the proportional gain, KP; and the integrator, KI. The parameters should be selected in this order.

To start, set the integrator to zero with the instruction

KI 0 <CR> Integrator gain

and set the proportional gain to a low value, such as

KP 1 <CR> Proportional gain

KD 100 <CR> Derivative gain

For more damping, you can increase KD (maximum is 4095). Increase gradually and stop after the motor vibrates. A vibration is noticed by audible sound or by interrogation. If you send the command

TE <CR> Tell error

a few times, and get varying responses, especially with reversing polarity, it indicates system vibration. When this happens, simply reduce KD.

Next you need to increase the value of KP gradually (maximum allowed is 1023). You can monitor the improvement in the response with the Tell Error instruction

KP 10 <CR> Proportion gain

TE <CR> Tell error

As the proportional gain is increased, the error decreases.

Again, the system may vibrate if the gain is too high. In this case, reduce KP. Typically, KP should not be greater than KD/4.

Finally, to select KI, start with zero value and increase it gradually. The integrator eliminates the position error, resulting in improved accuracy. Therefore, the response to the instruction

TE <CR>

becomes zero. As KI is increased, its effect is amplified and it may lead to vibrations. If this occurs, simply reduce KI.

For a more detailed description of the operation of the PID filter and/or servo system theory, see Chapter 10 Theory of Operation.

Step 10. Configure the Distributed Control System

The final step in Getting Started with the DMC-3425 distributed control system is to configure the individual controllers as their respective axes in the system. For more information on the operation of distributed control, please refer to Chapter 4.

Configuring Operation for Distributed Control

There are two methods for configuring a distributed control system; an automatic mode or a manual mode. The automatic mode uses a single command (HC) to configure all the slaves in a particular system. This command uses the BOOTP packets from the slaves, along with configuration jumpers, to automatically select IP addresses and set up the system. In the manual mode, slave controllers are assigned IP addresses and then configured into axes through various software commands. Both methods are outlined below.

Automatic Configuration of Distributed Control

The automatic method of assigning a distributed control network uses the HC command to indicate number of axes, type of communication and update rate of a system. This command also configures the number of IOC extended I/O modules in the system, if any.

The data update rate specifies the rate at which each slave sends a data packet to the master containing current status information. The data records are used by the master controller to make decisions based on the status of the slave controllers or IOC-7007 modules. This data record rate may be selected manually with the QW command, but will be set automatically by the second field of the HC command.

The data contained in the record is as follows:

reference position

encoder position

position error

velocity

torque

limit and home switches

axis status (in motion, motor off, at speed, stopcode)

uncommitted inputs

uncommitted outputs

user defined variables (4)

In order for the HC command to be initiated, an IP address must already be assigned to the master. See Step 5 “Establishing Communication between the DMC-3425 and the host PC” for information on addressing the master controller. The slaves, in this method, will typically remain without IP addresses. If the slaves are to be addressed manually while still using the HC mode, skip to the next section *Manual Slave IP configuration with HC command.*

Once initiated, the master controller will ARP for slaves with IP address already assigned, and then ‘listen’ for BOOTP packets from the slave controllers without IP addresses. As it receives these packets, the master will configure the slave axes according to jumpers set on each slave controller. Once this connection has been established, the master will initiate QW, or data records, to begin from each slave for status updates.

The full procedure for this method is as follows:

Step 1. Assign IP address to master controller either through IA command or through BOOTP utility in the Galil Software Registry. You may then burn this IP address into the master with the BN in order to keep this address during resets.

Step 2. Place jumpers on each slave controller indicating which slave corresponds to which axes in the system. See section "Step 2. Configuring Jumpers on the DMC-3425".

Step 3. Determine total number of axes, data update rate, and number of IOC-7007 controllers in the distributed system.

Step 4. Issue the command HCn,m,o,p where n is the total number of axes, m is the data update rate in milliseconds, o is a 1 for UDP communication or 2 for TCP/IP communication and p is the total number of IOC-7007's in the system. When using UDP communication, the HC command will assign one handle for both commands and QW records. When using TCP/IP communication, the HC command will assign one handle for commands and one handle for QW records. If o is a 3, then TCP/IP is used for commands, and UDP is used for QW records.

Step 5. Poll the operand _HC for success of connection. A response of 1 indicates the command is currently executing, a 2 for a successful configuration and a 0 for a failed configuration or no HC issued.

NOTE: The HC command may take up to 20 seconds to complete due to the time involved in waiting for the BOOTP packets.

Manual Slave IP configuration with HC command

It may be desired to manually assign an IP address to the slaves, while still using the HC command to connect to these slaves. This is possible, but you will need to take into account the addressing scheme the HC command is using, and you must install axis configuration jumpers according to "Step 2. Configuring jumpers on the DMC-3425".

When the HC command is initiated, the master will ARP addresses where it expects slave controllers to reside. If no controllers respond to the ARPs, the master will then 'listen' for the BOOTP packets from un-assigned slave controllers.

For addressing the slaves manually, the IP address MUST be assigned as follows. This will insure that the HC command will properly configure these controllers based on the master IP address.

Assume Master IP address = m.n.o.p where m, n, o and p is a valid Ethernet IP address.

First Slave IP address (Axis B or C) = m.n.o.p+2

Next slave is assigned +2 if previous slave was a single axis (DMC-3415).

Next slave is assigned +4 if previous slave was a dual axis (DMC-3425).

Slave axes are always assigned addresses based on their first axis.

IOC-7007 controllers are addressed as follows:

IOC 1 = m.n.o.p+16

IOC 2 = m.n.o.p+20

For example, in a 5 axis/1 IOC-7007 system with a DMC-3415 A axis Master, a DMC-3415 B axis, a DMC-3425 CD axis and a DMC-3415 E axis the following IP addresses would be set:

Assume Master IP address – 10.10.50.10

B Axis DMC-3415 – 10.10.50.12

CD Axis DMC-3425 – 10.10.50.14

E Axis DMC-3415 – 10.10.50.18

IOC-7007 (1) – 10.10.50.26

Automatic Configuration Example

The example below shows a typical setup file for the DMC-3425 distributed control system using the automatic configuration. This example is for a UDP system, with one handle used per slave. The IP addresses of the slaves are unassigned, as this is the simplest way for the slave controllers to be configured. The IP address of the master needs to have been assigned as described in Step 5 "Establishing Communication between the DMC-3425 and the host PC". The HC command will automatically assign those IP addresses based on the axis jumper settings described in Chapter 2.

Instruction	**Interpretation**
#SETUP	Begin Program
HC=6,20,1,0	Automatic configuration for a 6 axis UDP system with 20 msec update rate. The final 0 indicates no IOC-7007 Ethernet I/O modules in the system.
#LOOP; JP#LOOP,_HC=1	Wait while automatic configuration operates. This could take up to 10+ seconds.
IF (_HC=0)	Test for HC success. 0 = failed while 2 = success.
MG"CONFIGURATION FAILED"	
ELSE	
MG"CONFIG SUCCESS"	
ENDIF	
EN	

Manual Configuration of Distributed Control

For the manual configuration of distributed control, each 3425 must be assigned an IP address. This can be done with the BOOTP procedure in the Galil software or the IA command can be used to assign the IP address through the serial port. Once the IP address has been assigned, a BN command should be issued to save this value in the controller's non-volatile memory. Since all configuration is done manually in this method, there is no limit for the IP address of each slave in the system.

Upon power-up or reset, the master 3425 must establish each slave connection. The following steps must be taken while connected to the master 3425:

1. Using the IH command, open handles for each slave. For a TCP/IP connection, each slave controller must have 2 open handles, one for commands from the master, the other for data returned from the slave (QW). The second internet handle for each slave controller must contain a specific port value. The value must be an even number greater than 502. For a UDP connection, a slave controller can use a single handle for both commands from the master as well as data returned from the slave. The command for opening the communication handle is:

   IHh=ip0,ip1,ip2,ip3<p>n h is the handle. ip is the slave IP address. <p specifies port number. >n specifies connection type, 1 for UDP or 2 for TCP/IP.

2. Set the total number of axes in the system with the NA command. For example, assume there are 2 DMC-3425 slave cards, therefore there will be 6 axes (2 in the master and 4 in the slaves) and the command would be NA6.

3. Connect each slave handle to the master. This is accomplished with the CH command. The format of this command is:

   CHa=h1,h2 where a is the first axis designator of the slave controller, h1 is the handle for commands and h2 is the handle for slave status. h1 may equal h2 in a UDP setup.

Note that only one of the 2 axes (per DMC-3425) needs to be assigned with the CH command.

4. In order for the Master controller to be able to make decisions based on the status of the slave/server controllers, it is necessary for the slaves to generate data records giving their current status. The record is sent at a rate set by the QW command. The QW command must be executed by the master before the slave can issue a record under any method. The format of the command is

   QWh=n where h is the handle. n is a number between 4 and 16000.

   n sets the number of samples (msec with default TM1000).

   n equal to 0 disables the mode.

The data contained in the record is as follows:

reference position

encoder position

position error

velocity

torque

limit and home switches

axis status (in motion, motor off, at speed, stopcode)

uncommitted inputs

uncommitted outputs

user defined variables (4)

Manual Configuration Example

The example below shows a typical setup file for the DMC-3425 distributed control system in manual mode. This example is for a TCP/IP system, with two handles used per slave. The IP address of the first slave (Axes C and D) is 160.50.10.1, while the address of the second slave (Axes E and F) is 160.50.10.2. Note that in the two axis setup, different port numbers are used for the second handle to the same IP address.

Instruction	**Interpretation**
#SETUP	Begin Program
IHD=160,50,10,1>2	Set handle D (for commands) to slave 1's IP
IHE=160,50,10,1<510>2	Open handle E for slave 1's data record
IHF=160,50,10,2>2	Set handle F (for commands) to slave 2's IP
IHG=160,50,10,2<512>2	Open handle G for slave 2's data record
NA6	6 axis total
CHC=D,E	Axis C & D assigned to slave 1 (Handle D,E)
CHE=F,G	Axis E & F assigned to slave 2 (Handle F,G)
QWE=20	Handle E sends data record every 20 msec
QWG=20	Handle G sends data record every 20 msec
EN	

Note: This program is the minimum necessary for manually setting up the controller. An actual application program should make use of error and status checking. An example would be testing the operand _IHh2 for successful handle connections. See Command Reference for more details.

Design Examples

Here are a few examples for tuning and using your controller. These examples are shown for a single axis system only, but can be modified to test up to 8 axes within a distributed control network. See Chapter 6 Programming Motion for more examples of multi-axis programming.

Example 1 - System Set-up

This example assigns the system filter parameters, error limits and enables the automatic error shut-off.

Instruction	Interpretation
KP 10	Set proportional gain
KD 100	Set damping
KI 1	Set integral
OE 1	Set error off
ER 1000	Set error limit

Example 2 - Profiled Move

Objective: Rotate a distance of 10,000 counts at a slew speed of 20,000 counts/sec and an acceleration and deceleration rates of 100,000 counts/s^2.

Instruction	Interpretation
PR 10000	Distance
SP 20000	Speed
DC 100000	Deceleration
AC 100000	Acceleration
BGA	Start Motion

In response, the motor turns and stops.

Example 3 - Position Interrogation

The position of the A axis may be interrogated with the instruction

TPA	Tell position

which returns the position of the main encoder.

The position error, which is the difference between the commanded position and the actual position can be interrogated by the instructions

TEA	Tell error

Example 4 - Absolute Position

Objective: Command motion by specifying the absolute position.

Instruction	Interpretation
DP 0	Define the current position as 0
PA 7000	Sets the desired absolute position
BGA	Start motion on A axis

Example 5 - Velocity Control (Jogging)

Objective: Drive the motor at specified speeds.

Instruction	Interpretation
JG 10000	Set Jog Speed
AC 100000	Set acceleration
DC 50000	Set deceleration
BGA	Start motion on A axis

after a few seconds, command:

JG –40000	New speed and Direction
TVA	Returns speed

This causes velocity changes including direction reversal. The motion can be stopped with the instruction

STA	Stop

Example 6 - Operation Under Torque Limit

The magnitude of the motor command may be limited independently by the instruction TL. The following program illustrates that effect.

Instruction	Interpretation
TL 0.2	Set output limit to 0.2 volts
JG 10000	Set speed
BGA	Start motion on A axis

The motor will probably not move as the output signal is not sufficient to overcome the friction. If the motion starts, it can be stopped easily by a touch of a finger.

Increase the torque level gradually by instructions such as

TL 1.0	Increase torque limit to 1 volt.
TL 9.98	Increase torque limit to maximum, 9.98 Volts.

The maximum level of 10 volts provides the full output torque.

Example 7 - Interrogation

The values of the parameters may be interrogated using a ?. For example, the instruction

KP ?	Return gain

The same procedure applies to other parameters such as KI, KD, FA, etc.

Example 8 - Operation in the Buffer Mode

The instructions may be buffered before execution as shown below.

Instruction	Interpretation
PR 600000	Distance
SP 10000	Speed
WT 10000	Wait 10000 milliseconds before reading the next instruction
BGA	Start the motion

Example 9 - Motion Programs

Motion programs may be edited and stored in the memory. They may be executed at a later time.

The instruction

ED	Edit mode

moves the operation to the editor mode where the program may be written and edited. For example, in response to the first ED command, the Galil Windows software will open a simple editor window. From this window, the user can type in the following program:

#A	Define label
PR 700	Distance
SP 2000	Speed
BGA	Start motion
EN	End program

This program can be downloaded to the controller by selecting the File menu option download. Once this is done, close the editor.

Now the program may be executed with the command

XQ #A	Start the program running

Example 10 - Motion Programs with Loops

Motion programs may include conditional jumps as shown below.

Instruction	**Interpretation**
#A	Label
DP 0	Define current position as zero
V1=1000	Set initial value of V1
#Loop	Label for loop
PA V1	Move motor V1 counts
BGA	Start motion
AMA	After motion is complete
WT 500	Wait 500 ms
TPA	Tell position
V1=V1+1000	Increase the value of V1
JP #Loop,V1<10001	Repeat if V1<10001
EN	End

After the above program is entered, download the program from the File menu and exit the Editor. To start the motion, command:

XQ #A	Execute Program #A

Example 11- Motion Programs with Trippoints

The motion programs may include trippoints as shown below.

Instruction	**Interpretation**
#B	Label
DP0	Define initial position
PR 30000	Set target

SP 5000	Set speed
BGA	Start motion
AD 4000	Wait until A moved 4000
TPA	Tell position
EN	End program

To start the program, command:

XQ #B	Execute Program #B

Example 12 - Control Variables

Objective: To show how control variables may be utilized.

Instruction	Interpretation
#A;DP0	Label; Define current position as zero
PR 4000	Initial position
SP 2000	Set speed
BGA	Move
AMA	Wait until move is complete
WT 500	Wait 500 ms
#B	
V1 = _TP	Determine distance to zero
PR –V1/2	Command move 1/2 the distance
BGA	Start motion
AMA	After motion
WT 500	Wait 500 ms
V1=	Report the value of V1
JP #C, V1=0	Exit if position=0
JP #B	Repeat otherwise
#C;EN	End

To start the program, command

XQ #A	Execute Program #A

This program moves the motor to an initial position of 4000 and returns it to zero on increments of half the distance. Note, _TP is an internal variable that returns the value of the position. Internal variables may be created by preceding a DMC-3425 instruction with an underscore, _.

Example 13 - Control Variables and Offset

Objective: Illustrate the use of variables in iterative loops and use of multiple instructions on one line.

Instruction	Interpretation
#A	Set initial values
KI0	
DP0	
V1=8; V2=0	Initializing variables to be used by program
#B	Program label #B
OF V1	Set offset value
WT 200	Wait 200 msec

```
V2=_TP              Set variable V2 to the current position
JP#C,@ABS[V2]<2     Exit if error small
MG V2               Report value of V2
V1=V1-1             Decrease Offset
JP #B               Return to top of program
#C;EN               End
```

This program starts with a large offset and gradually decreases its value, resulting in decreasing error.

Chapter 3 Connecting Hardware

Overview

The DMC-3425 provides digital inputs for A and B **forward limit**, A and B **reverse limit**, A and B **home** input and **abort** input. The controller also has **3 uncommitted, TTL inputs, 3 TTL outputs** and **2 analog inputs (12-bit).**

The DMC-3415 provides a forward and reverse limit, home input and abort input. The controller also has **7 uncommitted, TTL inputs, 3 TTL outputs** and **2 analog inputs (12-bit).**

This chapter describes the inputs and outputs and their proper connection.

Using Inputs

Limit Switch Input

The forward limit switch (FLSx) inhibits motion in the forward direction immediately upon activation of the switch. The reverse limit switch (RLSx) inhibits motion in the reverse direction immediately upon activation of the switch. If a limit switch is activated during motion, the controller will make a decelerated stop using the deceleration rate previously set with the DC command. The motor will remain on (in a servo state) after the limit switch has been activated and will hold motor position. To set the activation state of the limit switches refer to the command CN, configure, in the Command Reference.

When a forward or reverse limit switch is activated, the current application program that is running will be interrupted and the controller will automatically jump to the #LIMSWI subroutine if one exists. This is a subroutine that the user can include in any motion control program and is useful for executing specific instructions upon activation of a limit switch.

After a limit switch has been activated, further motion in the direction of the limit switch will not be possible until the logic state of the switch returns back to an inactive state. This usually involves physically opening the tripped switch. Any attempt at further motion before the logic state has been reset will result in the following error: “022 - Begin not possible due to limit switch” error.

The operands, _LFx and _LRx, return the state of the forward and reverse limit switches, respectively (x represents the axis, A or B). The value of the operand is either a ‘0’ or ‘1’ corresponding to the logic state of the limit switch, active or inactive, respectively. If the limit switches are configured for active low, no connection or a 5V input will be read as a ‘0’, while grounding the switch will return a ‘1’. If the limit switches are configured for active high, the reading will be inverted and no connection or a 5V input will be read as a ‘1’, while grounding the switch will return a ‘0’.

Using a terminal program, the state of a limit switch can be printed to the screen with the command, MG _LFx or MG _LRx. This prints the value of the limit switch operands for the 'x' axis. The logic

state of the limit switches can also be interrogated with the TS command. For more details on TS, _LFx, _LRx, or MG see the Command Reference.

Home Switch Input

Homing inputs are designed to provide mechanical reference points for a motion control application. A transition in the state of a Home input alerts the controller that a particular reference point has been reached by a moving part in the motion control system. A reference point can be a point in space or an encoder index pulse.

The Home input detects any transition in the state of the switch and changes between logic states 0 and 1, corresponding to either 0V or 5V depending on the configuration set by the user (CN command). The CN command can be used to customize the homing routine to the user's application.

There are three homing routines supported by the DMC-3425: Find Edge (FE), Find Index (FI), and Standard Home (HM).

The Find Edge routine is initiated by the command sequence: FEx <return>, BGx <return> (where x could be any axis on the controller, A through H). The Find Edge routine will cause the motor to accelerate then slew at constant speed until a transition is detected in the logic state of the Home input. The direction of the FE motion is dependent on the state of the home switch. Refer to the CN command to set the correspondence between the Home Input voltage and motion direction. The motor will decelerate to a stop when a transition is seen on the input. The acceleration rate, deceleration rate and slew speed are specified by the user, prior to the movement, using the commands AC, DC, and SP. It is recommended that a high deceleration value be used so the motor will decelerate rapidly after sensing the Home switch.

The Find Index routine is initiated by the command sequence: FIx <return>, BGx <return> (where x could be any axis on the controller, A through H). Find Index will cause the motor to accelerate to the user-defined slew speed (SP) at a rate specified by the user with the AC command and slew until the controller senses a change in the index pulse signal from low to high. The motor then decelerates to a stop at the rate previously specified by the user with the DC command. Although Find Index is an option for homing, it is not dependent upon a transition in the logic state of the Home input, but instead is dependent upon a transition in the level of the index pulse signal.

The Standard Homing routine is initiated by the sequence of commands HMx <return>, BGx <return> (where x could be any axis on the controller, A through H). Standard Homing is a combination of Find Edge and Find Index homing. Initiating the standard homing routine will cause the motor to slew until a transition is detected in the logic state of the Home input. The motor will accelerate at the rate specified by the command, AC, up to the slew speed. After detecting the transition in the logic state on the Home Input, the motor will decelerate to a stop at the rate specified by the command DC. After the motor has decelerated to a stop, it switches direction and approaches the transition point at the speed of 256 counts/sec. When the logic state changes again, the motor moves forward (in the direction of increasing encoder count) at the same speed, until the controller senses the index pulse. After detection, it decelerates to a stop and defines this position as 0. The logic state of the Home input can be interrogated with the command MG _HMA. This command returns a 0 or 1 if the logic state is low or high (dependent on the CN command). The state of the Home input can also be interrogated indirectly with the TS command.

For examples and further information about Homing, see command HM, FI, FE of the Command Reference and the section entitled 'Homing' in the Programming Motion Section of this manual.

Abort Input

The function of the Abort input is to immediately stop the controller upon transition of the logic state.

NOTE: The response of the abort input is significantly different from the response of an activated limit switch. When the abort input is activated, the controller stops generating motion commands immediately, whereas the limit switch response causes the controller to make a decelerated stop.

NOTE: The effect of an Abort input is dependent on the state of the off-on-error function for each axis. If the Off-On-Error function is enabled for any given axis, the motor for that axis will be turned off when the abort signal is generated. This could cause the motor to 'coast' to a stop since it is no longer under servo control. If the Off-On-Error function is disabled, the motor will decelerate to a stop as fast as mechanically possible and the motor will remain in a servo state.

All motion programs that are currently running are terminated when a transition in the Abort input is detected. For information on setting the Off-On-Error function, see the Command Reference, OE.

Uncommitted Digital Inputs

The general use inputs are TTL and are accessible through the ICM-1460 or AMP-1460 as IN1 – IN3 for the DMC-3425 and IN1 – IN7 for the DMC-3415. The inputs can be accessed directly from the 37 Pin-D cable or connector on the controller, also. For a description of the pinouts, consult the appendix.

These inputs can be interrogated with the use of the command TI (Tell Inputs), the operand _TI, the function @IN[n] and the distributed I/O command TZ. All of these commands may be used locally to address individual controllers, or globally through the distributed control network. See Chapter 4 for a discussion of Global vs. Local communication as it pertains to I/O of the control system.

NOTE: For systems using the ICM-1460 or AMP-1460 interconnect module, there is an option to provide opto-isolation on the inputs. In this case, the user provides an isolated power supply (+5V to +24V and ground). For more information, see the section "Opto-Isolation Option for ICM-1460" in the Appendix of this manual, or consult Galil.

Amplifier Interface

The DMC-3425 analog command voltage, ACMD, ranges between +/-10V. This signal, along with GND, provides the input to the power amplifiers. The power amplifiers must be sized to drive the motors and load. For best performance, the amplifiers should be configured for a current mode of operation with no additional compensation. The gain should be set such that a 10 Volt input results in the maximum required current. If the controller is operating in stepper mode, the pulse and direction signals will be input into a stepper drive.

The DMC-3425 also provides an amplifier enable signal, AEN. This signal is activated under the following conditions: the watchdog timer activates, the motor-off command, MO, is given, or the OE1command (Enable Off-On-Error) is given and the position error exceeds the error limit. As shown in Figure 3.1, AEN can be used to disable the amplifier for these conditions.

Note: For a controller ordered as a DMC-3425-Stepper, the amplifier enable signal is used for the second stepper output.

The standard configuration of the AEN signal is TTL active high. In this configuration the AEN signal will be high when the controller expects the amplifier to be enabled. The polarity and the amplitude can be changed if you are using the ICM-1460 interface board. To change the polarity from active high (5 volts= enable, zero volts = disable) to active low (zero volts = enable, 5 volts= disable), replace the 7407 IC with a 7406. Note that many amplifiers designate the enable input as 'inhibit'.

To change the voltage level of the AEN signal, note the state of the jumper on the ICM/AMP-1460. When JP1 has a jumper from "AEN" to "5V" (default setting), the output voltage is 0-5V. To change to 12 volts, pull the jumper out and rotate it so that it connects the pins marked "AEN" and "+12V". If the jumper is removed entirely, the output is an open collector, allowing the user to connect an external supply with voltages up to 24V.

To connect an external 24V supply, remove the jumper JP1 from the interconnect board. Connect a 2.2kΩ resistor in series between the +24V of the supply and the amplifier enable terminal on the interconnect (AMPEN). Then wire the AMPEN to the enable pin on the amplifier. Connect the -24V

to the ground, GND, of the interconnect and connect the GND of the interconnect to the GND of the amplifier.

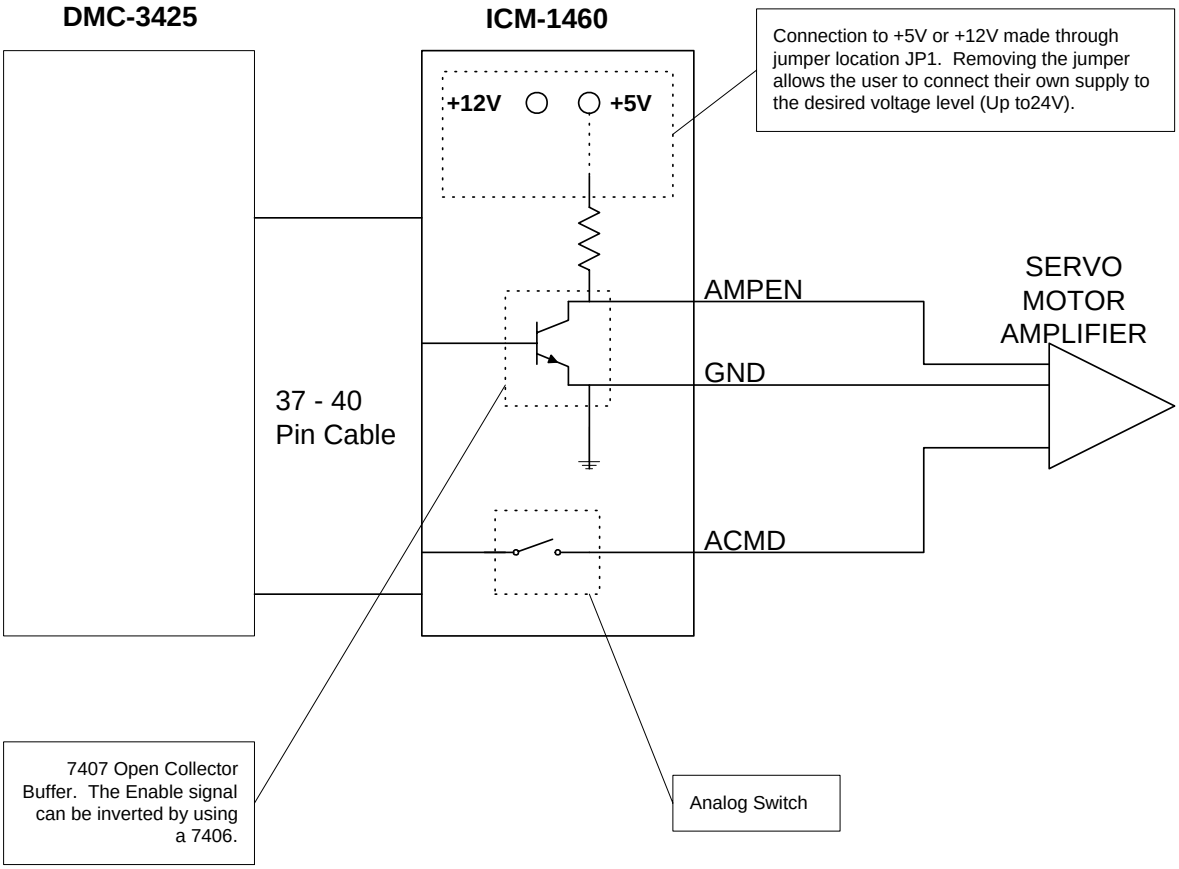


Figure 3.1 - Connecting AEN to the motor amplifier

TTL Inputs

As previously mentioned, the DMC-3425 has 3 uncommitted TTL level inputs while the DMC-3415 has 7 uncommitted TTL level inputs. The command @IN, TI and TZ will read the state of the inputs. For more information on these commands refer to the Command Reference.

The reset input is also a TTL level, non-isolated signal and is used to locally reset the DMC-3425 without resetting the PC.

Analog Inputs

The DMC-3425 has 2 analog inputs configured for the range between –10V and +10V. The inputs are decoded by a 12-bit ADC giving a voltage resolution of approximately .005V. The impedance of these inputs is 10Kohms. The analog inputs may be read using the @AN[n] function, where n is the number of the analog input to be read.

TTL Outputs

The DMC-3425 provides three general use outputs, an output compare and 4 status LED's.

The general use outputs are TTL and are accessible through the ICM-1460 as OUT1 thru OUT3. These outputs can be turned On and Off with the commands SB (Set Bit), CB (Clear Bit), OB (Output Bit) and OP (Output Port). For more information about these commands, see the Command Reference. The value of the outputs can be checked with the operand _OP, the function @OUT[] and the distributed control command TZ. Chapter 4 contains more information with regards to I/O in the distributed control network.

The output compare signal is TTL and is available on the ICM-1460 as CMP. Output compare is controlled by the position of any of the main encoders on the controller. The output can be programmed to produce an active low pulse (1usec) based on an incremental encoder value or to activate once when an axis position has been passed. For further information, see the command OC in the Command Reference.

Note: For a controller ordered as a DMC-3425-Stepper, the Error output is taken for the second stepper motor output.

There are four status LEDs on the controller, which indicate operating and error conditions on the controller. Below is a list of those LEDs and their functions.

Green Power LED - The green status LED indicates that the +5V power has been applied properly to the controller.

Red Status/Error LED - The red error LED will flash on initially at power up, and stay lit for approximately 1 – 8 seconds. After this initial power up condition, the LED will illuminate for the following reasons:

1. At least one axis has a position error greater than the error limit. The error limit is set by using the command ER.
2. The reset line on the controller is held low or is being affected by noise.
3. There is a failure on the controller and the processor is resetting itself.
4. There is a failure with the output IC which drives the error signal.

Green Link LED – The second green LED is lit when there is an Ethernet connection to the controller. This LED tests only for the physical connection, not for an active or enabled link.

Yellow Activity LED – The yellow LED indicates traffic across the Ethernet connection. This LED will show both transmit and receive activity across the connection. If there is no Ethernet connection or IP address assigned, the LED will flash at regular intervals to show that the BOOTP packets are being broadcast.

Note: For systems using the ICM-1460 or AMP-1460 interconnect module, there is an option to provide opto-isolation for the outputs. In this case, the user provides an isolated power supply (+5V to +24V and ground). For more information, see the section "Opto-Isolation Option for ICM-1460" in the Appendix of this manual, or contact Galil.

THIS PAGE LEFT BLANK INTENTIONALLY

Chapter 4 Communication

Introduction

The DMC-3425 has one RS232 port and one Ethernet port. The RS-232 port is the data set. The Ethernet port is a 10Base-T link. The RS-232 is a standard serial link with communication baud rates up to 19.2kbaud.

For initial setup, Galil recommends starting with the RS-232 interface. The RS-232 provides a simplified interface that minimizes the potential problems for first time setup. Once the configuration parameters have been properly set and saved on the controller, the Ethernet communication should be established.

RS232 Port

The DMC-3425 has a single RS232 connection for sending and receiving commands from a PC or other terminal. The pin-outs for the RS232 connection are as follows.

RS232 - Port 1 DATATERM

1 CTS – output
2 Transmit Data - output
3 Receive Data - input
4 RTS – input
5 Ground
6 CTS – output
7 RTS – input
8 CTS – output
9 No connect (Can connect to +5V or sample clock)

RS-232 Configuration

Configure your PC for 8-bit data, one start-bit, one stop-bit, full duplex and no parity. The baud rate for the RS232 communication can be selected by selecting the proper jumper configuration on the DMC-3425 according to the table below.

Baud Rate Selection

JUMPER SETTINGS		BAUD RATE
96	**12**	**--**
OFF	OFF	19200
ON	OFF	9600

OFF	ON	1200

Handshaking Modes

The RS232 port is configured for hardware handshaking. In this mode, the RTS and CTS lines are used. The CTS line will go high whenever the DMC-3425 is not ready to receive additional characters. The RTS line will inhibit the DMC-3425 from sending additional characters. Note: The RTS line goes high for inhibit. This handshake procedure ensures proper communication especially at higher baud rates.

Ethernet Configuration

Communication Protocols

The Ethernet is a local area network through which information is transferred in units known as packets. Communication protocols are necessary to dictate how these packets are sent and received. The DMC-3425 supports two industry standard protocols, TCP/IP and UDP/IP. The controller will automatically respond in the format in which it is contacted.

TCP/IP is a "connection" protocol. The master must be connected to the slave in order to begin communicating. Each packet sent is acknowledged when received. If no acknowledgement is received, the information is assumed lost and is resent.

Unlike TCP/IP, UDP does not require a "connection". This protocol is similar to communicating via RS232. If information is lost, the controller does not return a colon or question mark. Because the protocol does not provide for lost information, the sender must re-send the packet.

Ethernet communication transfers information in ‘packets’. The packets must be limited to 470 data bytes or less. Larger packets could cause the controller to lose communication.

NOTE: In order not to lose information in transit, Galil recommends that the user wait for an acknowledgement of receipt of a packet before sending the next packet.

Addressing

There are three levels of addresses that define Ethernet devices. The first is the Ethernet or hardware address. This is a unique and permanent 6 byte number. No other device will have the same Ethernet address. The DMC-3425 Ethernet address is set by the factory and the last two bytes of the address are the serial number of the controller.

The second level of addressing is the IP address. This is a 32-bit (or 4 byte) number. The IP address is constrained by each local network and must be assigned locally. Assigning an IP address to the controller can be done in a number of ways.

The first method is to use the BOOT-P utility via the Ethernet connection (the DMC-3425 must be connected to network and powered). For a brief explanation of BOOT-P, see the section: *Third Party Software*. Either a BOOT-P server on the internal network or the Galil terminal software may be used. To use the Galil BOOT-P utility, select the registry in the terminal emulator. Next, select the DMC-3425 controller communicating via Ethernet from the software registry. Once the controller has been selected, the next screen shows options for the actual connection. Enter the IP address at the prompt and select either TCP/IP or UDP/IP as the protocol. When done, click on the ASSIGN IP ADDRESS. The Galil Terminal Software will respond with a list of all controllers on the network that do not currently have IP addresses. The user selects the controller and the software will assign the controller the specified IP address. Then enter the terminal and type in BN to save the IP address to the controller's non-volatile memory. A full description of addressing the card may be found in Chapter 2 Getting Started.

CAUTION: Be sure that there is only one BOOT-P server running. If your network has DHCP or BOOT-P running, it may automatically assign an IP address to the controller upon linking it to the network. In order to ensure that the IP address is correct, please contact your system administrator before connecting the controller to the Ethernet network.

The second method for setting an IP address is to send the IA command through the DMC-3425 main RS-232 port. The IP address you want to assign may be entered as a 4 byte number delimited by commas (industry standard uses periods) or a signed 32 bit number. (Ex. IA 124,51,29,31 or IA 2083724575) Type in BN to save the IP address to the controller's non-volatile memory.

NOTE: Galil strongly recommends that the IP address selected is not one that can be accessed across the Gateway. The Gateway is an application that controls communication between an internal network and the outside world.

The third level of Ethernet addressing is the UDP or TCP port number. The Galil controller does not require a specific port number. The port number is established by the client or master each time it connects to the controller.

Ethernet Handles

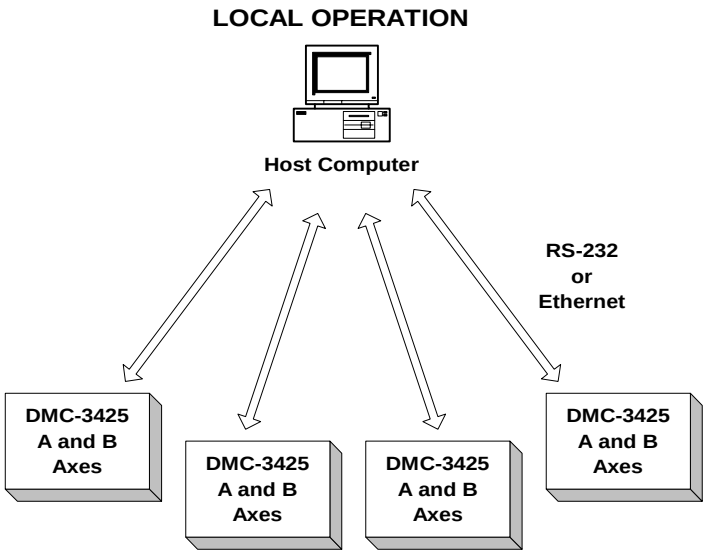
An Ethernet handle is a communication resource within a device. The DMC-3425 can have a maximum of 8 Ethernet handles open at any time. When using TCP/IP, each connection to a device, such as the host computer, requires an individual Ethernet handle. In UDP/IP, one handle may be used for all the masters, but each slave uses one. (Pings and ARP's do not occupy handles.) If all 8 handles are in use and a 9th master tries to connect, it will be sent a "reset packet" that generates the appropriate error in its windows application.

The TH command may be used to indicate which handles are currently connected to and which are currently free.

Global vs. Local Operation

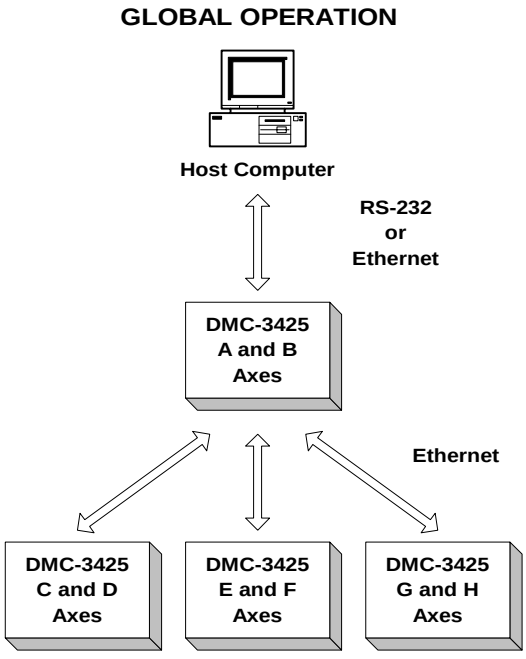
Each DMC-3425 controls two axes of motion, referred to as A and B. The host computer can communicate directly with any DMC-3425 using an Ethernet or RS-232 connection. When the host computer is directly communicating with any DMC-3425, all commands refer to the first two axes as A and B. Direct communication with the DMC-3425 is known as LOCAL OPERATION.

The concept of Local and Global Operation also applies to application programming. See Chapter 7: Global vs. Local Programming.



The DMC-3425 supports Galil's Distributed Control System. This allows up to 4 DMC-3425s to be connected together as a single virtual 8-axis controller. In this system, one of the controllers is designated as the master. The master can receive commands from the host computer that apply to all of the axes in the system.

A simple way to view Local and Global Operation: When the host communicates with a slave controller, it considers the slave as a 2-axis controller. When the host communicates with a master, it considers the master as a multi-axis controller. Similarly, an application program residing in a slave controller deals only with 2 motors as A & B. An application program in a master deals with all motors referenced as A through H.



The controllers may operate under both Local and/or Global Mode. In general, operating in Global Mode simplifies controlling the entire system. However, Local Mode operation is necessary in some

situations; using Local Mode for setup and testing is useful since this isolates the controller. Specific modes of motion require operation in Local Mode. Also, each controller can have a program, including the slave controllers. When a slave controller has a program, this program would always operate in Local Mode.

Operation of Distributed Control

For most commands it is not necessary to be conscious of whether an axis is local or remote. For instance to set the KP value for the A and C axes, the command to the master would be

```
KP 10,,20
```

Similarly, the interrogation commands can also be issued. For example, the position error for all axes would be TE. The position operand for the F axis would be_TPF.

Some commands inherently are sent to all controllers. These include commands such as AB (abort), CN and TM. In addition, the * may be used to send commands to all controllers. For example

```
SP*=1000
```

will send a speed of 1000 cts/sec to all axes. This syntax may be used with any configuration or parameter commands.

Certain commands need to be launched specifically. For this purpose there is the SA command. In its simplest form the SA command is

```
SAh= "command string"
```

Here "command string" will be sent to handle h. For example, the SA command is the means for sending an XQ command to a slave/server. A more flexible form of the command is

```
SAh= field1,field2,field3,field4 ... field8
```

where each field can be a string in quotes or a variable.

For example, to send the command KI,,5,10; Assume var1=5 and var2=10 and send the command:

```
SAF= "KI",var1,var2
```

When the Master/client sends an SA command to a Slave/server, it is possible for the master to determine the status of the command. The response _IHh4 will return the number 1 to 4. One means waiting for the acknowledgement from the slave. Two means a colon (command accepted) has been received. Three means a question mark (command rejected) has been received. Four means the command timed out.

If a command generates responses (such as the TE command), the values will be stored in _SAh0 thru _SAh7. If a field is unused its _SA value will be -2^31.

Accessing the I/O of the Slaves

The I/O of the server/slaves is settable and readable from the master. The bit numbers are adjusted by the handle number of the slave controller. Each handle adds 100 to the bit number. Handle A is 100 and handle H is 800. In a TCP/IP control setup with two handles per slave, Galil recommends using the value of the first handle for simplicity. In a UDP system, the single handle per slave is used to address the I/O.

The command TZ can be used to display all of the digital I/O contained in a distributed control system. Any IOC-7007's configured using the HC command will also be displayed with the TZ command. See the Command Reference for more information on the TZ command.

Digital Outputs

For outputs, the SB and CB commands are used to command individual output ports, while the OP command is used for setting bytes of data. The SB and CB commands may be set globally through the master, while the OP command must be sent to the slave using the SA command.

Outputs may be set globally according to the following numbering scheme: Bitnum = (Slave Handle * 100) + Output Bit. For example:

Set Bit 2 on a UDP distributed slave using the E handle for communication. The E handle would have a numerical value of 500, plus the bit number of 2. The command would therefore become SB502.

Specific outputs in a distributed system may be read by using the @OUT[n] function, where n is the corresponding bit number as defined above.

Output bits on an IOC-7007 may also be set through the master controller in a distributed network. Please refer to the IOC-7007 Manual for information on setting and reading these I/O points.

Digital Inputs

Digital inputs may be addressed individually using the @IN[n] function, or in blocks using the TI command. Both of these commands may be sent globally to the controller. The 'n' in the @IN[n] function operates identically to the SB/CB syntax. This means that a specific input bit is referenced as the slave handle number * 100 plus the input bit. For example:

Read input bit 4 on a TCP/IP distributed slave using the C handle for communication. The C handle in this case would give a value of 300. Therefore, to read bit 4, the command would be MG@IN[304]. The MG in this case simply displays this data to the terminal.

The TI command may be used to read all inputs on a slave in blocks of 8. This is helpful if the slave controller in question has a DB-14064 expanded I/O daughter card. The TI command uses the slave handle number * 100 plus the block number to be read. The block number is only used if the controller has the DB-14064 expansion option.

Inputs on an IOC-7007 may also be read through the master controller in a distributed network. Please refer to the IOC-7007 Manual for information on setting and reading these points.

Analog Inputs

Each DMC-3425 controller has two 12-bit analog inputs. These inputs are read with the command @AN[n], where n is the input to be read. The master controller has n = 1 and 2, the first slave controller uses n = 3 and 4, etc.

Handling Communication Errors

A new automatic subroutine which is identified by the label #TCPERR, has been added. If a controller has an application program running and the TCP or UDP communication is lost, the #TCPERR routine will automatically execute. The #TCPERR routine should be ended with a RE command. In the UDP configuration, the QW commands must be active in order for the #TCPERR routine on the master to operate properly.

Multicasting

A multicast may only be used in UDP and is similar to a broadcast, (where everyone on the network gets the information) but specific to a group. In other words, all devices within a specified group will receive the information that is sent in a multicast. There can be many multicast groups on a network and are differentiated by their multicast IP address. To communicate with all the devices in a specific multicast group, the information can be sent to the multicast IP address rather than to each individual device IP address. All Galil controllers belong to a default multicast address of 239.255.19.56. The controller's multicast IP address can be changed by using the IA> u command.

The Galil Registry has an option to disable the opening of the multicast handle on the DMC-3425. By default this multicast handle will be opened.

Unsolicited Message Handling

Anytime a controller generates an internal response from a program, generates an internal error or sends a message from a program using the MG command, this is termed an unsolicited message. There are two software commands that will configure how the controller handles these messages; the CW and the CF command.

The DMC-3425 has 8 Ethernet handles as well as 1 serial port where unsolicited messages may be sent. The CF command is used to configure the controller to send these messages to specific ports. In addition, the Galil Registry has various options for sending this CF command. For more information, see the CF command in the DMC-3425 Command Reference. The MG can also send the message to a specific handle using the MG{Eh} syntax, where h is the handle. See the MG command in the Command Reference for more information.

The CW command has two data fields that affect unsolicited messages. The first field configures the most significant bit (MSB) of the message. A value of 1 will set the MSB of unsolicited messages, while a value of 2 suppresses the MSB. The majority of software programs use a setting of CW2, although the Galil Smart Terminal and WSDK will set this to CW1 for internal usage. If you have difficulty receiving characters from the controller, or receive garbage characters instead of messages, check the status of the CW command for a setting of CW2.

IOC-7007 Support

The IOC-7007 is an Intelligent Ethernet I/O controller that can be programmed in standard Galil language. This module allows various configurations of TTL inputs, opto-isolated inputs, high power outputs and relay switches to be used in the Galil distributed motion system. Each IOC-7007 may be populated by up to seven IOM I/O modules.

The IOC-7007 Ethernet I/O controller may be used in a distributed system and commanded by the master controller. The HC command is used to specify total number of IOC-7007 controllers within that distributed system. Once configured, the I/O of that IOC-7007 becomes incorporated in the distributed system, much the same as board level I/O of the DMC-3425 slaves.

Inputs of the IOC-7007 are read using the standard @IN[n] and TI commands as follows:

@IN[n] where n is the IOC-7007 input bit to be read. n is calculated with the equation n = (HandleNum * 1000) + BitNum. HandleNum is the numeric value of the IOC-7007 handle (1 – 8) while BitNum is the specific bit number on the IOC to be read.

TIn where n is the IOC-7007 input slot to be read. n is calculated with the equation n = (HandleNum * 1000) + SlotNum. Again, HandleNum is the numeric value of the IOC-7007 handle (1 – 8). SlotNum corresponds to the location of the IOM input module in the 7 slots of the IOC-7007 (0 – 6). This will return either an 8 bit or 16 bit decimal value depending on which IOM input module is being used.

Outputs of the IOC-7007 are set and cleared using the standard SB and CB commands, as well as with the OQ and OB commands. Outputs can be read with the @OUT[n] command. These commands operate as follows:

SBn or CBn where n is the IOC-7007 output to be set or cleared. n is calculated identically to the @IN[n] configuration, with n = (HandleNum * 1000) + BitNum.

@OUT[n] where n is the IOC-7007 output to be read. This uses the same n configuration as SB and CB.

OQn,m where n is the IOC-7007 output location and m is the data to be written. Specifically, n = (HandleNum * 1000) + SlotNum where HandleNum is the numeric value of the IOC-7007 handle

(1 – 8) and SlotNum is the slot number of the IOM output module to be written to (0 – 6). m is the decimal representation of the data written to the 4 (0 – 15) or 8 (0 – 255) output points of the IOM module.

Please refer to the IOC-7007 manual for complete information on how to configure, read and write information to the IOC-7007 Ethernet I/O module.

Modbus Support

The Modbus protocol supports communication between masters and slaves. The masters may be multiple PC's that send commands to the controller. The slaves are typically peripheral I/O devices that receive commands from the controller.

When the Galil controller acts as the master, the IH command is used to assign handles and connect to its slaves. The IP address may be entered as a 4 byte number separated with commas (industry standard uses periods) or as a signed 32 bit number. A port number may also be specified, and should be set to 502, which is the Modbus defined port number. The protocol (TCP/IP or UDP/IP) to use must also be designated at this time. Otherwise, the controller will not connect to the slave. (Ex. IHB=151,25,255,9<502>2 - This will open handle #2 and connect to the IP address 151.25.255.9, port 502, using TCP/IP)

An additional protocol layer is available for speaking to I/O devices. Modbus is an RS-485 protocol that packages information in binary packets that are sent as part of a TCP/IP packet. In this protocol, each slave has a 1 byte slave address. The DMC-3425 can use a specific slave address or default to the handle number.

The Modbus protocol has a set of commands called function codes. The DMC-3425 supports the 10 major function codes:

Function Code	Definition
01	Read Coil Status (Read Bits)
02	Read Input Status (Read Bits)
03	Read Holding Registers (Read Words)
04	Read Input Registers (Read Words)
05	Force Single Coil (Write One Bit)
06	Preset Single Register (Write One Word)
07	Read Exception Status (Read Error Code)
15	Force Multiple Coils (Write Multiple Bits)
16	Preset Multiple Registers (Write Words)
17	Report Slave ID

The DMC-3425 provides three levels of Modbus communication. The first level allows the user to create a raw packet and receive raw data. It uses the MBh command with a function code of –1. The format of the command is

MBh = -1,len,array[] where len is the number of bytes

array[] is the array with the data

The second level incorporates the Modbus structure. This is necessary for sending configuration and special commands to an I/O device. The formats vary depending on the function code that is called. For more information refer to the Command Reference.

The third level of Modbus communication uses standard Galil commands. Once the slave has been configured, the commands that may be used are @IN[], @AN[], SB, CB, OB, and AO. For example, AO 2020,8.2 would tell I/O number 2020 to output 8.2 volts.

If a specific slave address is not necessary, the I/O number to be used can be calculated with the following:

I/O Number = (HandleNum*1000) +((Module-1)*4) + (BitNum-1)

Where HandleNum is the handle number from 1 (A) to 8 (H). Module is the position of the module in the rack from 1 to 16. BitNum is the I/O point in the module from 1 to 4.

If an explicit slave address is to be used, the equation becomes:

I/O Number = (SlaveAddress*10000) + (HandleNum*1000) +((Module-1)*4) + (Bitnum-1)

To view an example procedure for communicating with an OPTO-22 rack, refer to the appendix.

Other Communication Options

User Defined Ethernet Variables

It may be necessary within a distributed system to share information that is not contained as position, torque, velocity or other control data. The DMC-3425 provides 2 user defined variables that are passed as part of the QW record shared among the distributed system. In this way, it is not necessary for a single controller to write variable data directly to all the other controllers in the system.

ZA and ZB are two user defined variables which are passed with the QW record at each update. Data that is written to these variables is then seen by the master DMC-3425 in the system.

Handle Switching

By default, when initiating a communication session with a DMC-3425 controller, the first available handle is used. If no handles have been assigned to the controller, the A handle is chosen. The command HS allows the user to switch this connection to another handle, freeing up the initial handle or trading with another currently used handle. Or, once handles have been defined, the HS command may be used to switch handles to prioritize slave locations and I/O locations.

Handle Restore on Communication Failure

There are instances within an Ethernet system, whether UDP or TCP/IP, when a handle may become disconnected without closing properly. An example of this would be a simple cable failure, where the Ethernet cable of a certain slave becomes detached.

The command HR is used to enable a mode in which the master controller, upon seeing a failure on a handle, will attempt to restore that handle. This is helpful when a distributed system is already fully configured and a slave is lost. The #TCPERR routine can be used to flag the error, while the handle restore will attempt to reconnect to the slave until the problem is fixed. This makes it unnecessary to re-run the setup for the entire distributed system.

Note: This function is only available if the system has been configured using the automatic handle configuration command, HC.

Waiting on Handle Responses

The operation of the distributed network has commands being sent to the master controller, which then distributes these commands to the slave axes in the system. For example, the command PR10,10,10,10,10,10,10,10 sent to the master becomes packets of PR10,10 sent by the master to each of the slaves in the system. When the slave receives this command from the master, a colon or question mark is generated and sent back to the master to acknowledge the command.

The HW command allows the user to select whether or not the master will wait on this colon response from the slave. If the HW is set to 0, the master will not wait for these responses. This results in faster command execution but could cause problems if any slave errors are generated. The setting HW1, on the other hand, insures that the master knows of any slave errors but does result in a slightly increased command execution time as it waits for these responses.

Data Record

The DMC-3425 can provide a block of status information with the use of a single command, QR. This command, along with the QZ command can be very useful for accessing complete controller status. The QR command will return 4 bytes of header information and specific blocks of information as specified by the command arguments: QR ABCDEFGHS

Each argument corresponds to a block of information according to the Data Record Map below. If no argument is given, the entire data record map will be returned. Note that the data record size will depend on the number of axes.

NOTE: A, B, C, & D can be interchanged with X, Y, Z, & W respectively.

Data Record Map

DATA TYPE	ITEM	BLOCK
UB	1st byte of header	Header
UB	2nd byte of header	Header
UB	3rd byte of header	Header
UB	4rth byte of header	Header
UW	sample number	I block
UB	general input bank 0 (Inputs 1-7)	I block
UB	general input bank 1 (Always 0)	I block
UB	general input bank 2 (DB-14064)	I block
UB	general input bank 3 (DB-14064)	I block
UB	general input bank 4 (DB-14064)	I block
UB	general input bank 5 (DB-14064)	I block
UB	general input bank 6 (DB-14064)	I block
UB	general input bank 7 (DB-14064)	I block
UB	general input bank 8 (DB-14064)	I block
UB	general input bank 9 (DB-14064)	I block
UB	general output bank 0 (Outputs 1 – 3)	I block
UB	general output bank 1 (Always 0)	I block

UB	general output bank 2 (DB-14064)	I block
UB	general output bank 3 (DB-14064)	I block
UB	general output bank 4 (DB-14064)	I block
UB	general output bank 5 (DB-14064)	I block
UB	general output bank 6 (DB-14064)	I block
UB	general output bank 7 (DB-14064)	I block
UB	general output bank 8 (DB-14064)	I block
UB	general output bank 9 (DB-14064)	I block
UB	error code	I block
UB	general status	I block
UW	segment count of coordinated move for S plane	S block
UW	coordinated move status for S plane	S block
SL	distance traveled in coordinated move for S plane	S block
UW	0	T block
UW	0	T block
SL	0	T block
UW	A axis status	A block
UB	A axis switches	A block
UB	A axis stopcode	A block
SL	A axis reference position	A block
SL	A axis motor position	A block
SL	A axis position error	A block
SL	A axis auxiliary position	A block
SL	A axis velocity	A block
SW	A axis torque	A block
SW	Analog Input 1	A block
UW	B axis status	B block
UB	B axis switches	B block
UB	B axis stopcode	B block
SL	B axis reference position	B block
SL	B axis motor position	B block
SL	B axis position error	B block
SL	B axis auxiliary position	B block
SL	B axis velocity	B block
SW	B axis torque	B block
SW	Analog Input 2	B block
UW	C axis status	C block
UB	C axis switches	C block
UB	C axis stopcode	C block
SL	C axis reference position	C block
SL	C axis motor position	C block
SL	C axis position error	C block
SL	C axis auxiliary position	C block
SL	C axis velocity	C block
SW	C axis torque	C block

SW	C axis analog input	C block
UW	D axis status	D block
UB	D axis switches	D block
UB	D axis stopcode	D block
SL	D axis reference position	D block
SL	D axis motor position	D block
SL	D axis position error	D block
SL	D axis auxiliary position	D block
SL	D axis velocity	D block
SW	D axis torque	D block
SW	D axis analog input	D block
UW	E axis status	E block
UB	E axis switches	E block
UB	E axis stopcode	E block
SL	E axis reference position	E block
SL	E axis motor position	E block
SL	E axis position error	E block
SL	E axis auxiliary position	E block
SL	E axis velocity	E block
SW	E axis torque	E block
SW	E axis analog input	E block
UW	F axis status	F block
UB	F axis switches	F block
UB	F axis stopcode	F block
SL	F axis reference position	F block
SL	F axis motor position	F block
SL	F axis position error	F block
SL	F axis auxiliary position	F block
SL	F axis velocity	F block
SW	F axis torque	F block
SW	F axis analog input	F block
UW	G axis status	G block
UB	G axis switches	G block
UB	G axis stopcode	G block
SL	G axis reference position	G block
SL	G axis motor position	G block
SL	G axis position error	G block
SL	G axis auxiliary position	G block
SL	G axis velocity	G block
SW	G axis torque	G block
SW	G axis analog input	G block
UW	H axis status	H block
UB	H axis switches	H block
UB	H axis stopcode	H block
SL	H axis reference position	H block

SL	H axis motor position	H block
SL	H axis position error	H block
SL	H axis auxiliary position	H block
SL	H axis velocity	H block
SW	H axis torque	H block
SW	H axis analog input	H block

NOTE: UB = Unsigned Byte, UW = Unsigned Word, SW = Signed Word, SL = Signed Long Word

Explanation of Status Information and Axis Switch Information

Header Information - Byte 0, 1 of Header:

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
1	N/A	N/A	N/A	N/A	I Block Present in Data Record	T Block Present in Data Record	S Block Present in Data Record
BIT 7	**BIT 6**	**BIT 5**	**BIT 4**	**BIT 3**	**BIT 2**	**BIT 1**	**BIT 0**
H Block Present in Data Record	G Block Present in Data Record	F Block Present in Data Record	E Block Present in Data Record	D Block Present in Data Record	C Block Present in Data Record	B Block Present in Data Record	A Block Present in Data Record

Bytes 2, 3 of Header:

Bytes 2 and 3 make a word that represents the Number of bytes in the data record, including the header. Byte 2 is the low byte and byte 3 is the high byte

NOTE: The header information of the data records is formatted in little endian.

General Status Information (1 Byte)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Program Running	N/A	N/A	N/A	N/A	Waiting for input from IN command	Trace On	Echo On

Axis Switch Information (1 Byte)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Latch Occurred	State of Latch Input	N/A	N/A	State of Forward Limit	State of Reverse Limit	State of Home Input	SM Jumper Installed

Axis Status Information (2 Byte)

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
Move in Progress	Mode of Motion PA or PR	Mode of Motion PA only	(FE) Find Edge in Progress	Home (HM) in Progress	1st Phase of HM complete	2nd Phase of HM complete or FI command issued	Mode of Motion Coord. Motion
BIT 7	**BIT 6**	**BIT 5**	**BIT 4**	**BIT 3**	**BIT 2**	**BIT 1**	**BIT 0**
Negative Direction Move	Mode of Motion Contour	Motion is slewing	Motion is stopping due to ST or Limit Switch	Motion is making final decel.	Latch is armed	Off-On-Error occurred	Motor Off

Coordinated Motion Status Information for plane (2 Byte)

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
Move in Progress	N/A	N/A	N/A	N/A	N/A	N/A	N/A
BIT 7	**BIT 6**	**BIT 5**	**BIT 4**	**BIT 3**	**BIT 2**	**BIT 1**	**BIT 0**
N/A	N/A	Motion is slewing	Motion is stopping due to ST or Limit Switch	Motion is making final decel.	N/A	N/A	N/A

Notes Regarding Velocity and Torque Information

The velocity information that is returned in the data record is 64 times larger than the value returned when using the command TV (Tell Velocity). See command reference for more information about TV.

The Torque information is represented as a number in the range of +/-32767. Maximum negative torque is -32767. Maximum positive torque is 32767. Zero torque is 0.

QZ Command

The QZ command can be very useful when using the QR command, since it provides information about the controller and the data record. The QZ command returns the following 4 bytes of information.

BYTE #	INFORMATION
0	Number of axes present
1	Number of bytes in general block of data record
2	Number of bytes in coordinate plane block of data record
3	Number of Bytes in each axis block of data record

Using Third Party Software

Galil supports ARP, BOOT-P, and Ping, which are utilities for establishing Ethernet connections. ARP is an application that determines the Ethernet (hardware) address of a device at a specific IP address. BOOT-P is an application that determines which devices on the network do not have an IP address and assigns the IP address you have chosen to it. Ping is used to check the communication between the device at a specific IP address and the host computer.

The DMC-3425 can communicate with a host computer through any application that can send TCP/IP or UDP/IP packets. A good example of this is Telnet, a utility that comes with most Windows systems. In the absence of the Galil Windows Terminal software, the Telnet terminal may be used for communication with the DMC-3425 Ethernet controller. The Windows Hyperterminal may also be used for communication.

THIS PAGE LEFT BLANK INTENTIONALLY

Chapter 5 Command Basics

Introduction

The DMC-3425 provides over 100 commands for specifying motion and machine parameters. Commands are included to initiate action, interrogate status and configure the digital filter. These commands can be sent in ASCII or binary.

In ASCII, the DMC-3425 instruction set is BASIC-like and easy to use. Instructions consist of two uppercase letters that correspond phonetically with the appropriate function. For example, the instruction BG begins motion, and ST stops the motion. In binary, commands are represented by a binary code ranging from 80 to FF.

ASCII commands can be sent "live" over the bus for immediate execution by the DMC-3425, or an entire group of commands can be downloaded into the DMC-3425 memory for execution at a later time. Combining commands into groups for later execution is referred to as Applications Programming and is discussed in the following chapter. Binary commands cannot be used in Applications programming.

This section describes the DMC-3425 instruction set and syntax. A summary of commands as well as a complete listing of all DMC-3425 instructions is included in the *Command Reference* chapter.

Command Syntax - ASCII

DMC-3425 instructions are represented by two ASCII upper case characters followed by applicable arguments. A space may be inserted between the instruction and arguments. A semicolon or <return> is used to terminate the instruction for processing by the DMC-3425 command interpreter.

NOTE: If you are using a Galil terminal program, commands will not be processed until a <return> command is given. This allows the user to separate many commands on a single line and not begin execution until the user gives the <return> command.

IMPORTANT: All DMC-3425 commands are sent in upper case.

For example, the command

PR 4000 <return> Position relative

PR is the two character instruction for position relative. 4000 is the argument which represents the required position value in counts. The <return> terminates the instruction. The space between PR and 4000 is optional.

For specifying data for the A,B,C and D axes, commas are used to separate the axes. If no data is specified for an axis, a comma is still needed as shown in the examples below. If no data is specified for an axis, the previous value is maintained.

To view the current values for each command, type the command followed by a ? for each axis requested.

PR 1000	Specify A only as 1000
PR ,2000	Specify B only as 2000
PR ,,3000	Specify C only as 3000
PR ,,,4000	Specify D only as 4000
PR 2000, 4000,6000, 8000	Specify A B C and D
PR ,8000,,9000	Specify B and D only
PR ?,?,?,?	Request A,B,C,D values
PR ,?	Request B value only

The DMC-3425 provides an alternative method for specifying data. Here data is specified individually using a single axis specifier such as A,B,C or D. An equals sign is used to assign data to that axis. For example:

PRA=1000	Specify a position relative movement for the A axis of 1000
ACB=200000	Specify acceleration for the B axis as 200000

Instead of data, some commands request action to occur on an axis or group of axes. For example, ST AB stops motion on both the A and B axes. Commas are not required in this case since the particular axis is specified by the appropriate letter A, B, C or D. If no parameters follow the instruction, action will take place on all axes. Here are some examples of syntax for requesting action:

BG A	Begin A only
BG B	Begin N only
BG ABCD	Begin all axes
BG BD	Begin B and D only
BG	Begin all axes

For controllers with 5 or more axes, the axes are referred to as A,B,C,D,E,F,G,H.

BG ABCDEFGH	Begin all axes
BG D	Begin D only

Coordinated Motion with more than 1 axis

When requesting action for coordinated motion, the letter S is used to specify a coordinated motion plane. For example:

BG S	Begin coordinated sequence, S
BG SW	Begin coordinated sequence, S, and W axis

Command Syntax - Binary

Some commands have an equivalent binary value. Binary communication mode can be executed much faster than ASCII commands. Binary format can only be used when commands are sent from the PC and cannot be embedded in an application program.

Binary Command Format

All binary commands have a 4 byte header and are followed by data fields. The 4 bytes are specified in hexadecimal format.

Header Format:

Byte 1

Specifies the command number between 80 to FF. The complete binary command number table is listed below.

Byte 2

Specifies the # of bytes in each field as 0,1,2,4 or 6 as follows:

00	No datafields (i.e. SH or BG)
01	One byte per field
02	One word (2 bytes per field)
04	One long word (4 bytes) per field
06	Galil real format (4 bytes integer and 2 bytes fraction)

Byte 3

Specifies whether the command applies to a coordinated move as follows:

00	No coordinated motion movement
01	Coordinated motion movement

For example, the command STS designates motion to stop on a vector motion. The third byte for the equivalent binary command would be 01.

Byte 4

Specifies the axis # or data field as follows

Bit 7 = H axis or 8^{th} data field

Bit 6 = G axis or 7^{th} data field

Bit 5 = F axis or 6^{th} data field

Bit 4 = E axis or 5^{th} data field

Bit 3 = D axis or 4^{th} data field

Bit 2 = C axis or 3^{rd} data field

Bit 1 = B axis or 2^{nd} data field

Bit 0 = A axis or 1^{st} data field

Datafields Format

Datafields must be consistent with the format byte and the axes byte. For example, the command PR 1000,, -500 would be

A7 02 00 05 03 E8 FE 0C

where A7 is the command number for PR

02 specifies 2 bytes for each data field

00 S is not active for PR

05 specifies bit 0 is active for A axis and bit 2 is active for C axis ($2^0 + 2^2=5$)

03 E8 represents 1000

FE OE represents -500

Example

The command ST ABCS would be

A1 00 01 07

where A1 is the command number for ST

00 specifies 0 data fields

01 specifies stop the coordinated axes S

07 specifies stop A (bit 0), B (bit 1) and C (bit 2) $2^0+2^1+2^3 =7$

Binary command table

COMMAND	NO.	COMMAND	NO.	COMMAND	No.
Reserved	80	reserved	ab	reserved	d6
KP	81	reserved	ac	reserved	d7
KI	82	reserved	ad	RP	d8
KD	83	reserved	ae	TP	d9
DV	84	reserved	af	TE	da
AF	85	LM	b0	TD	db
KF	86	LI	b1	TV	dc
PL	87	VP	b2	RL	dd
ER	88	CR	a3	TT	de
IL	89	TN	b4	TS	df
TL	8a	LE, VE	b5	TI	e0
MT	8b	VT	b6	SC	e1
CE	8c	VA	b7	reserved	e2
OE	8d	VD	b8	reserved	e3
FL	8e	VS	b9	reserved	e4
BL	8f	VR	ba	TM	e5
AC	90	reserved	bb	CN	e6
DC	91	reserved	bc	LZ	e7
SP	92	CM	bd	OP	e8

IT	93	CD	be	OB	e9
FA	94	DT	bf	SB	ea
FV	95	ET	c0	CB	eb
GR	96	EM	c1	I I	ec
DP	97	EP	c2	EI	ed
DE	98	EG	c3	AL	ee
OF	99	EB	c4	reserved	ef
GM	9a	EQ	c5	reserved	f0
Reserved	9b	EC	c6	reserved	f1
Reserved	9c	reserved	c7	reserved	f2
Reserved	9d	AM	c8	reserved	f3
Reserved	9e	MC	c9	reserved	f4
Reserved	9f	TW	ca	reserved	f5
BG	a0	MF	cb	reserved	f6
ST	a1	MR	cc	reserved	f7
AB	a2	AD	cd	reserved	f8
HM	a3	AP	ce	reserved	f9
FE	a4	AR	cf	reserved	fa
FI	a5	AS	d0	reserved	fb
PA	a6	AI	d1	reserved	fc
PR	a7	AT	d2	reserved	fd
JG	a8	WT	d3	reserved	fe
MO	a9	WC	d4	reserved	ff
SH	aa	reserved	d5		

Controller Response to DATA

The DMC-3425 returns a : for valid commands and a ? for invalid commands.

For example, if the command BG is sent in lower case, the DMC-3425 will return a ?.

:bg <return>	invalid command, lower case
?	DMC-3425 returns a ?

When the controller receives an invalid command the user can request the error code. The error code will specify the reason for the invalid command response. To request the error code, type the command: TC1. For example:

?TC1 <return>	Tell Code command
1 Unrecognized command	Returned response

There are many reasons for receiving an invalid command response. The most common reasons are: unrecognized command (such as typographical entry or lower case), command given at improper time (such as during motion), or a command out of range (such as exceeding maximum speed). A complete listing of all codes is listed in the TC command in the Command Reference section.

Interrogating the Controller

Interrogation Commands

The DMC-3425 has a set of commands that directly interrogate the controller. When the command is entered, the requested data is returned in decimal format on the next line followed by a carriage return and line feed. The format of the returned data can be changed using the Position Format (PF), Variable Format (VF) and Leading Zeros (LZ) command. See Chapter 7 and the Command Reference.

Summary of Interrogation Commands

RP	Report Command Position
RL	Report Latch
^R ^V	Firmware Revision Information
SC	Stop Code
TB	Tell Status
TC	Tell Error Code
TD	Tell Dual Encoder
TE	Tell Error
TI	Tell Input
TP	Tell Position
TR	Trace
TS	Tell Switches
TT	Tell Torque
TV	Tell Velocity

For example, the following example illustrates how to display the current position of the A axis:

TP A <return>	Tell position A
0000000000	Controllers Response
TP AB <return>	Tell position A and B
0000000000,0000000000	Controllers Response

Interrogating Current Commanded Values.

Most commands can be interrogated by using a question mark (?) as the axis specifier. Type the command followed by a ? for each axis requested.

PR ?,?,?,?	Request A,B,C,D values
PR ,?	Request B value only

The controller can also be interrogated with operands.

Operands

Most DMC-3425 commands have corresponding operands that can be used for interrogation. Operands must be used inside of valid DMC expressions. For example, to display the value of an operand, the user could use the command:

MG ‘operand’ where ‘operand’ is a valid DMC operand

All of the command operands begin with the underscore character (_). For example, the value of the current position on the A axis can be assigned to the variable ‘V’ with the command:

V=_TPA

The Command Reference denotes all commands which have an equivalent operand as "Used as an Operand". Also, see description of operands in Chapter 7.

Command Summary

For a complete command summary, see *Command Reference* manual.

THIS PAGE LEFT BLANK INTENTIONALLY

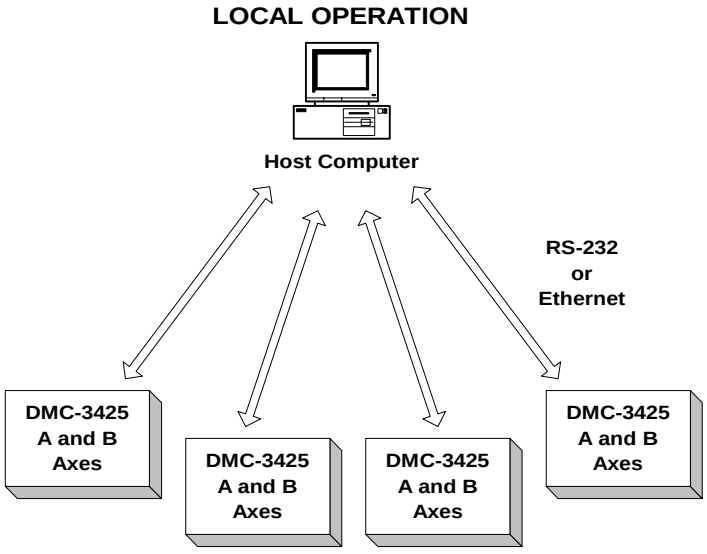
Chapter 6 Programming Motion

Overview

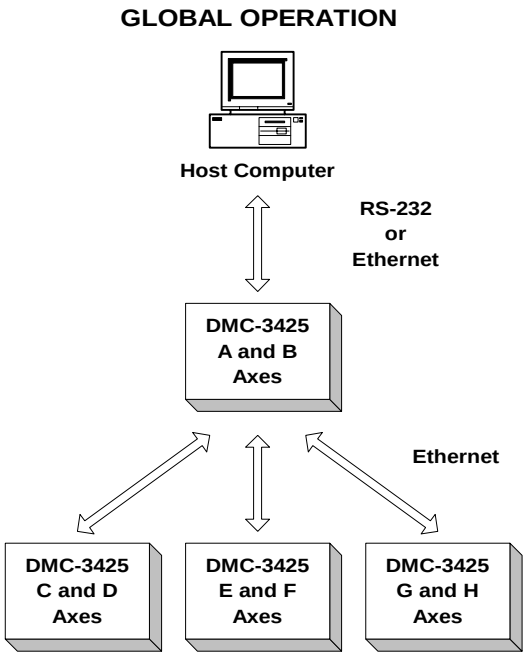
The DMC-3425 provides many modes of motion, including independent positioning and jogging, coordinated motion, electronic cam motion, and electronic gearing. Each one of these modes is discussed in the following sections.

Global vs. Local Operation

Each DMC-3425 controls two axes of motion, referred to as A and B. The host computer can communicate directly with any DMC-3425 using an Ethernet or RS-232 connection. When the host computer is directly communicating with any DMC-3425, all commands refer to the first two axes as A and B. Direct communication with the DMC-3425 is known as LOCAL OPERATION.



The DMC-3425 supports Galil's Distributed Control System. This allows up to eight axes of DMC-3425 and DMC-3415 controllers to be connected together as a single virtual axis controller. In this system, one of the controllers is designated as the master. The master can receive commands from the host computer that apply to all of the axes in the system.



The controllers may operate under both Local and/or Global mode. In general, operating in Global mode simplifies controlling the entire system. However, Local Mode operation is necessary in some situations; Using local mode for setup and testing is useful since this isolates the controller. Specific modes of motion require operation in Local Mode. Also, each controller can have a program, including the slave controllers. When a slave controller has a program, this program would always operate in Local mode.

The following table describes the modes of motion and whether this mode will work in Global or Local Mode:

Mode of Motion	Basic description	Commands	Global	LOCAL
Relative Independent Axis Positioning	Each axis operates independently and motion is specified with a relative distance, velocity, acceleration and deceleration. The axis follows the prescribed velocity profile.	PR, AC, DC, SP	YES	YES
Absolute Independent Axis Positioning	Each axis operates independently and motion is specified with an absolute position, velocity, acceleration and deceleration. The axis follows the prescribed velocity profile.	PA, AC, DC, SP	YES	YES
Independent Jogging	Each axis operates independently and the axis follows a prescribed velocity profile with no final endpoint. The motion is specified with velocity, acceleration and deceleration. Motion stops on Stop command.	JG AC, DC ST	YES	YES
Linear Interpolation	2 thru 8 axes of coordinated motion. The path is described by linear incremental segments and vector velocity, vector acceleration and vector deceleration. The vector motion follows the prescribed velocity profile.	LM LI, LE VS, VA, VD	NO	YES

Vector Motion	2-D motion path consisting of arc segments and linear segments, such as engraving or quilting. Vector velocity, vector acceleration and vector deceleration are specified. The vector motion follows the prescribed velocity profile.	VM VP, CR VS, VA, VD	NO	YES
Electronic Gearing	Motion in which one axis must follow another axis such as conveyer speed. Once setup, the slave axis will follow the master position.	GA GR	NO	YES
Contour Mode	1 – 8 axes of motion along arbitrary profiles or mathematically prescribed profiles such as sine or cosine trajectories. The path is described by linear incremental segments and the time between segments	CM CD DT	NO	YES
Electronic Cam	Following a trajectory based on a master encoder position.	EA EM EP ET	NO	YES

Independent Axis Positioning

In this mode, motion between the specified axes is independent, and each axis follows its own profile. The user specifies the desired absolute position (PA) or relative position (PR), slew speed (SP), acceleration ramp (AC), and deceleration ramp (DC), for each axis. On begin (BG), the DMC-3425 profiler generates the corresponding trapezoidal or triangular velocity profile and position trajectory. The controller determines a new command position along the trajectory every sample period until the specified profile is complete. Motion is complete when the last position command is sent by the DMC-3425 profiler.

NOTE: The actual motor motion may not be complete when the profile has been completed, however, the next motion command may be specified.

The Begin (BG) command can be issued for all axes either simultaneously or independently. ABC or D axis specifiers are required to select the axes for motion. When no axes are specified, this causes motion to begin on all axes.

The speed (SP) and the acceleration (AC) can be changed at any time during motion; however, the deceleration (DC) and position (PR or PA) cannot be changed until motion is complete. Remember, motion is complete when the profiler is finished, not when the actual motor is in position. The Stop command (ST) can be issued at any time to decelerate the motor to a stop before it reaches its final position.

An incremental position movement (IP) may be specified during motion as long as the additional move is in the same direction. Here, the user specifies the desired position increment, n. The new target is equal to the old target plus the increment, n. Upon receiving the IP command, a revised profile will be generated for motion towards the new end position. The IP command does not require a BG.

NOTE: If the motor is not moving, the IP command is equivalent to the PR and BG command combination.

Command Summary - Independent Axis

COMMAND	DESCRIPTION
PR a,b,c,d	Specifies relative distance
PA a,b,c,d	Specifies absolute position
SP a,b,c,d	Specifies slew speed
AC a,b,c,d	Specifies acceleration rate
DC a,b,c,d	Specifies deceleration rate
BG ABCD	Starts motion
ST ABCD	Stops motion before end of move
IP a,b,c,d	Changes position target
IT a,b,c,d	Time constant for independent motion smoothing
AM ABCD	Trippoint for profiler complete
MC ABCD	Trippoint for "in position"

The lower case specifiers (a,b,c,d) represent position values for each axis.

The DMC-3425 also allows use of single axis specifiers such as PRA=2000

Operand Summary - Independent Axis

OPERAND	DESCRIPTION
_ACn	Return acceleration rate for the axis specified by ‘n
_DCn	Return deceleration rate for the axis specified by ‘n’
_SPn	Returns the speed for the axis specified by ‘n’
_PAn	Returns current destination if ‘n’ axis is moving, otherwise returns the current commanded position if not in a move.
_PRn	Returns current incremental distance specified for the ‘n’ axis

Examples

Absolute Position Movement

Instruction	Interpretation
PA 10000,20000	Specify absolute A,B position
AC 1000000,1000000	Acceleration for A,B
DC 1000000,1000000	Deceleration for A,B
SP 50000,30000	Speeds for A,B
BG AB	Begin motion

Multiple Move Sequence

Required Motion Profiles:

A-Axis	1000 counts	Position
	15000 count/sec	Speed
	500000 counts/sec^2	Acceleration/Deceleration
B-Axis	500 counts	Position

	10000 count/sec	Speed
	500000 counts/sec^2	Acceleration/Deceleration
C-Axis	100 counts	Position
	5000 counts/sec	Speed
	500000 counts/sec	Acceleration/Deceleration

This example will specify a relative position movement on A, B and C axes. The movement on each axis will be separated by 20 msec. Fig. 6.1 shows the velocity profiles for the A,B and C axis.

Instruction	**Interpretation**
#A	Begin Program
PR 1000,500,100	Specify relative position movement of 1000, 500 and 100 counts for A,B and C axes.
SP 15000,10000,5000	Specify speed of 10000, 15000, and 5000 counts / sec
AC 500000,500000,500000	Specify acceleration of 500000 counts / sec^2 for all axes
DC 500000,500000,500000	Specify deceleration of 500000 counts / sec^2 for all axes
BG A	Begin motion on the A axis
WT 20	Wait 20 msec
BG B	Begin motion on the B axis
WT 20	Wait 20 msec
BG C	Begin motion on C axis
EN	End Program

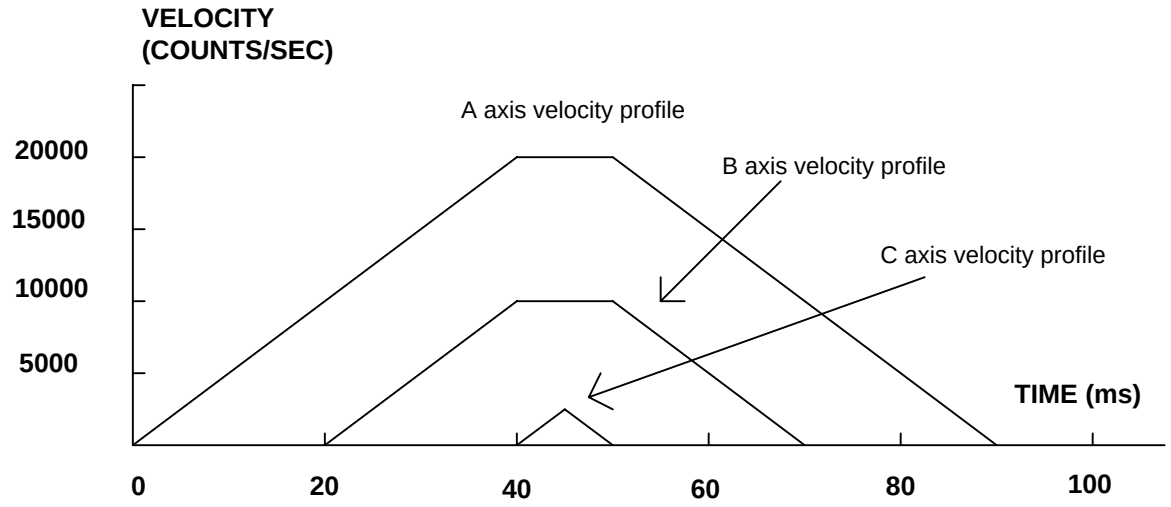


Figure 6.1 - Velocity Profiles of ABC

Notes on fig 6.1: The A and B axis have a 'trapezoidal' velocity profile, while the C axis has a 'triangular' velocity profile. The A and B axes accelerate to the specified speed, move at this constant speed, and then decelerate such that the final position agrees with the command position, PR. The C axis accelerates, but before the specified speed is achieved, must begin deceleration such that the axis will stop at the commanded position. All 3 axes have the same acceleration and deceleration rate, hence, the slope of the rising and falling edges of all 3 velocity profiles are the same.

Independent Jogging

The jog mode of motion is very flexible because speed, direction and acceleration can be changed during motion. The user specifies the jog speed (JG), acceleration (AC), and the deceleration (DC) rate for each axis. The direction of motion is specified by the sign of the JG parameters. When the begin command is given (BG), the motor accelerates up to speed and continues to jog at that speed until a new speed or stop (ST) command is issued. If the jog speed is changed during motion, the controller will make a accelerated (or decelerated) change to the new speed.

An instant change to the motor position can be made with the use of the IP command. Upon receiving this command, the controller commands the motor to a position which is equal to the specified increment plus the current position. This command is useful when trying to synchronize the position of two motors while they are moving.

Note that the controller operates as a closed-loop position controller while in the jog mode. The DMC-3425 converts the velocity profile into a position trajectory and a new position target is generated every sample period. This method of control results in precise speed regulation with phase lock accuracy.

Command Summary - Jogging

COMMAND	DESCRIPTION
AC a,b,c,d	Specifies acceleration rate
BG ABCD	Begins motion
DC a,b,c,d	Specifies deceleration rate
IP a,b,c,d	Increments position instantly
IT a,b,c,d	Time constant for independent motion smoothing
JG +/- a,b,c,d	Specifies jog speed and direction
ST ABCD	Stops motion

Parameters can be set with individual axes specifiers such as JGB=2000 (set jog speed for B axis to 2000) or ACBH=400000 (set acceleration for B and H axes to 400000).

Operand Summary - Independent Axis

OPERAND	DESCRIPTION
_ACn	Return acceleration rate for the axis specified by ‘n’
_DCn	Return deceleration rate for the axis specified by ‘n’
_SPn	Returns the jog speed for the axis specified by ‘n’
_TVn	Returns the actual velocity of the axis specified by ‘n’ (averaged over .25 sec)

Examples

Jog in A and C axes

Jog A motor at 50000 count/s. After A motor is at its jog speed, begin jogging C in reverse direction at 25000 count/s.

Instruction	Interpretation
#A	Label
AC 20000,,20000	Specify A,C acceleration of 20000 cts / sec
DC 20000,,20000	Specify A,C deceleration of 20000 cts / sec
JG 50000,,-25000	Specify jog speed and direction for A and C axis
BG A	Begin A motion
AS A	Wait until A is at speed
BG C	Begin C motion
EN	

Joystick Jogging

The jog speed can also be changed using an analog input such as a joystick. Assume that for a 10 Volt input the speed must be 50000 counts/sec.

Instruction	Interpretation
#JOY	Label
JG0	Set in Jog Mode
BGA	Begin motion
#B	Label for loop
V1 =@AN[1]	Read analog input
VEL=V1*50000/10	Compute speed
JG VEL	Change JG speed
JP #B	Loop

Linear Interpolation Mode (Local Mode)

The DMC-3425 provides a linear interpolation mode for 2 axes. In linear interpolation mode, motion between the axes is coordinated to maintain the prescribed vector speed, acceleration, and deceleration along the specified path. The motion path is described in terms of incremental distances for each axis. An unlimited number of incremental segments may be given in a continuous move sequence, making the linear interpolation mode ideal for following a piece-wise linear path. There is no limit to the total move length.

The LM command selects the Linear Interpolation mode and axes for interpolation. Since the DMC3425 is a 2-axis controller, the LM command would specify LM AB.

When using the linear interpolation mode, the LM command only needs to be specified once unless the axes for linear interpolation change.

Specifying Linear Segments

The command LI x,y or LI a,b specifies the incremental move distance for each axis. This means motion is prescribed with respect to the current axis position. Up to 511 incremental move segments may be given prior to the Begin Sequence (BGS) command. Once motion has begun, additional LI segments may be sent to the controller.

The clear sequence (CS) command can be used to remove LI segments stored in the buffer prior to the start of the motion. To stop the motion, use the instructions STS or AB. The command, ST, causes a decelerated stop. The command, AB, causes an instantaneous stop and aborts the program, and the command AB1 aborts the motion only.

The Linear End (LE) command must be used to specify the end of a linear move sequence. This command tells the controller to decelerate to a stop following the last LI command. If an LE command is not given, an Abort AB1 must be used to abort the motion sequence.

It is the responsibility of the user to keep enough LI segments in the DMC-3425 sequence buffer to ensure continuous motion. If the controller receives no additional LI segments and no LE command, the controller will stop motion instantly at the last vector. There will be no controlled deceleration. LM? or _LM returns the available spaces for LI segments that can be sent to the buffer. 511 returned means the buffer is empty and 511 LI segments can be sent. A zero means the buffer is full and no additional segments can be sent. As long as the buffer is not full, additional LI segments can be sent at PC bus speeds.

The instruction _CS returns the segment counter. As the segments are processed, _CS increases, starting at zero. This function allows the host computer to determine which segment is being processed.

Additional Commands

The commands VS n, VA n, and VD n are used to specify the vector speed, acceleration and deceleration. The vector speed is computed using the equation:

$VS^2=AS^2+BS^2$, where AS, and BS are the speed of the A, and B axes.

The controller always uses the axis specifications from LM, not LI, to compute the speed.

VT is used to set the S-curve smoothing constant for coordinated moves. The command AV n is the 'After Vector' trippoint, which halts program execution until the vector distance of n has been reached.

Specifying Vector Speed for Each Segment

The instruction VS has an immediate effect and, therefore, must be given at the required time. In some applications, such as CNC, it is necessary to attach various speeds to different motion segments. This can be done by two functions: < n and > m

For example: LI x,y < n >m

The first command, < n, is equivalent to commanding VSn at the start of the given segment and will cause an acceleration toward the new commanded speeds, subjects to the other constraints.

The second function, > m, requires the vector speed to reach the value m at the end of the segment. Note that the function > m may start the deceleration within the given segment or during previous segments, as needed to meet the final speed requirement, under the given values of VA and VD.

Note, however, that the controller works with one > m command at a time. As a consequence, one function may be masked by another. For example, if the function >100000 is followed by >5000, and the distance for deceleration is not sufficient, the second condition will not be met. The controller will attempt to lower the speed to 5000, but will reach that at a different point.

As an example, consider the following program.

Instruction	Interpretation
#ALT	Label for alternative program
DP 0,0	Define Position of A and B axis to be 0
LMAB	Define linear mode between A and B axes.
LI 4000,0 <4000 >1000	Specify first linear segment with a vector speed of 4000 and end speed 1000
LI 1000,1000 < 4000 >1000	Specify second linear segment with a vector speed of 4000 and end speed 1000
LI 0,5000 < 4000 >1000	Specify third linear segment with a vector speed of 4000 and end speed 1000

LE	End linear segments
BGS	Begin motion sequence
EN	Program end

Changing Feedrate:

The command VR n allows the feedrate, VS, to be scaled between 0 and 10 with a resolution of .0001. This command takes effect immediately and causes VS to be scaled. VR also applies when the vector speed is specified with the '<' operator. This is a useful feature for feedrate override. VR does not ratio the accelerations. For example, VR .5 results in the specification VS 2000 to be divided in half.

Command Summary - Linear Interpolation

COMMAND	DESCRIPTION
LM nn	Specify axes for linear interpolation
LM ?	Returns number of available spaces for linear segments in DMC-3425 sequence buffer. Zero means buffer full. 512 means buffer empty.
LI x,y < n LI a,b < n	Specify incremental distances relative to current position, and assign vector speed n.
VS n	Specify vector speed
VA n	Specify vector acceleration
VD n	Specify vector deceleration
VR n	Specify the vector speed ratio
BGS	Begin Linear Sequence
CS	Clear sequence
LE	Linear End- Required at end of LI command sequence
LE?	Returns the length of the vector (resets after 2147483647)
AMS	Trippoint for After Sequence complete
AV n	Trippoint for After Relative Vector distance, n
VT	S curve smoothing constant for vector moves

Operand Summary - Linear Interpolation

OPERAND	DESCRIPTION
_AV	Return distance traveled
_CS	Segment counter - returns number of the segment in the sequence, starting at zero.
_LE	Returns length of vector (resets after 2147483647)
_LM	Returns number of available spaces for linear segments in DMC-3425 sequence buffer. Zero means buffer full. 512 means buffer empty.
_VPm	Return the absolute coordinate of the last data point along the trajectory. (m=A,B)

To illustrate the ability to interrogate the motion status, consider the first motion segment of our example, #LMOVE, where the A axis moves toward the point A=5000. Suppose that when A=3000, the controller is interrogated using the command 'MG _AV'. The returned value will be 3000. The value of _CS, _VPA and _VPB will be zero.

Now suppose that the interrogation is repeated at the second segment when Y=2000. The value of _AV at this point is 7000, _CS equals 1, _VPA=5000 and _VPB=0.

Example

Linear Interpolation Motion

In this example, the AB system is required to perform a 90° turn. In order to slow the speed around the corner, we use the AV 4000 trippoint, which slows the speed to 1000 count/s. Once the motors reach the corner, the speed is increased back to 4000 cts / s.

Instruction	Interpretation
#LMOVE	Label
DP 0,0	Define position of A and B axes to be 0
LMAB	Define linear mode between A and B axes.
LI 5000,0	Specify first linear segment
LI 0,5000	Specify second linear segment
LE	End linear segments
VS 4000	Specify vector speed
BGS	Begin motion sequence
AV 4000	Set trippoint to wait until vector distance of 4000 is reached
VS 1000	Change vector speed
AV 5000	Set trippoint to wait until vector distance of 5000 is reached
VS 4000	Change vector speed
EN	Program end

Example - Linear Move

Make a coordinated linear move in the AB plane. Move to coordinates 40000,30000 counts at a vector speed of 100000 counts/sec and vector acceleration of 1000000 counts/sec^2.

Instruction	Interpretation
LM AB	Specify axes for linear interpolation
LI40000,30000	Specify AB distances
LE	Specify end move
VS 100000	Specify vector speed
VA 1000000	Specify vector acceleration
VD 1000000	Specify vector deceleration
BGS	Begin sequence

Note that the above program specifies the vector speed, VS, and not the actual axis speeds VA and VB the axis speeds are determined by the DMC-3425 from:

$$VS = \sqrt{VA^2 + VB^2}$$

The resulting profile is shown in Figure 6.2.

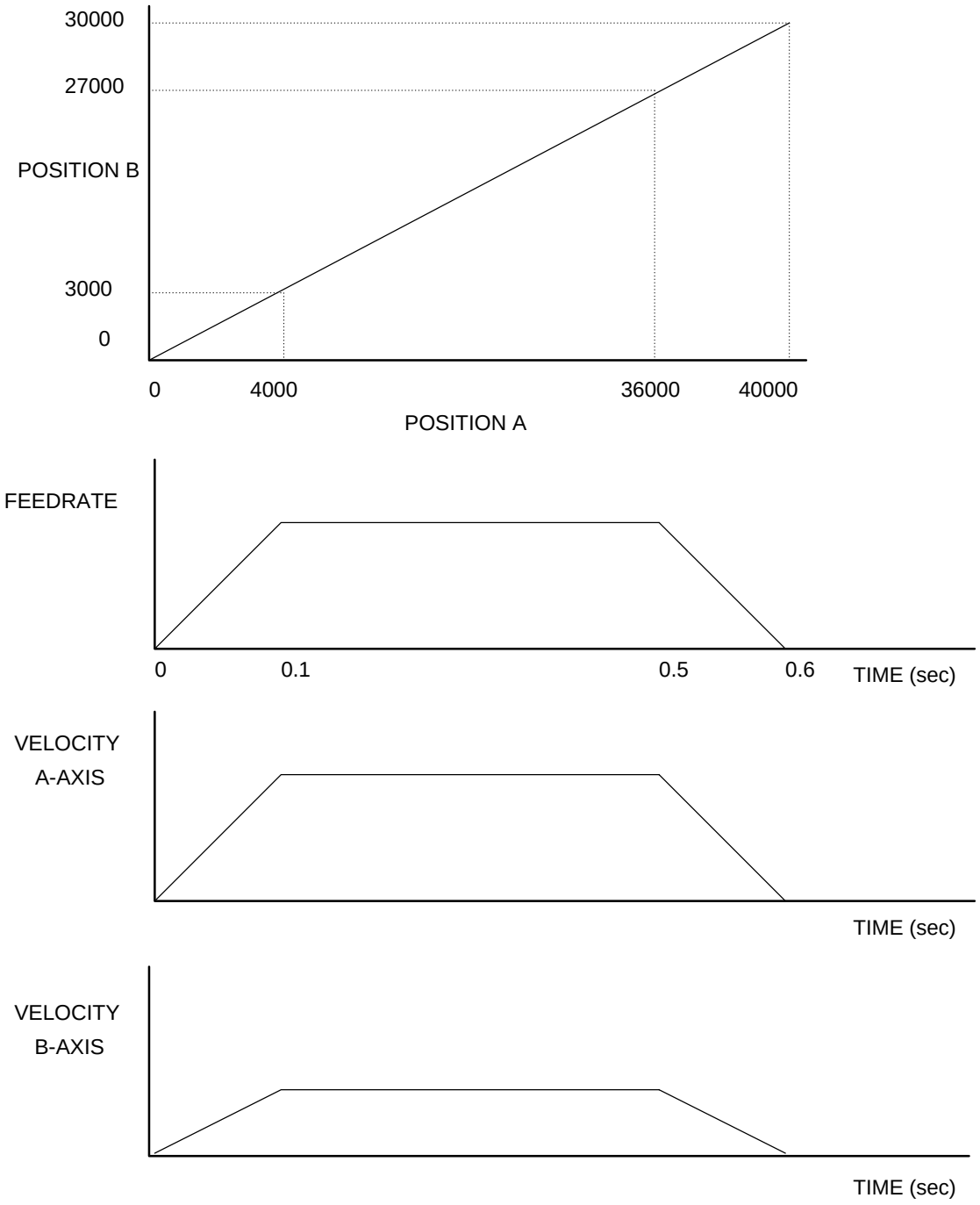


Figure 6.2 - Linear Interpolation

Example - Multiple Moves

This example makes a coordinated linear move in the AB plane. The Arrays VA and VB are used to store 750 incremental distances which are filled by the program #LOAD.

Instruction	**Interpretation**
#LOAD	Load Program
DM VA [750],VB [750]	Define Array

```
COUNT=0                          Initialize Counter
N=10                             Initialize position increment
#LOOP                            LOOP
VA [COUNT]=N                     Fill Array VA
VB [COUNT]=N                     Fill Array VB
N=N+10                           Increment position
COUNT=COUNT+1                    Increment counter
JP #LOOP,COUNT<750               Loop if array not full
#A                               Label
LM AB                            Specify linear mode for AB
COUNT=0                          Initialize array counter
#LOOP2;JP#LOOP2,_LM=0            If sequence buffer full, wait
JS#C,COUNT=500                   Begin motion on 500th segment
LI VA[COUNT],VB[COUNT]           Specify linear segment
COUNT=COUNT+1                    Increment array counter
JP #LOOP2,COUNT<750              Repeat until array done
LE                               End Linear Move
AMS                              After Move sequence done
MG "DONE"                        Send Message
EN                               End program
#C;BGS;EN                        Begin Motion Subroutine
```

Vector Mode: Linear and Circular Interpolation (Local Mode)

The DMC-3425 allows a long 2-D path consisting of linear and arc segments to be prescribed. Motion along the path is continuous at the chosen vector speed even at transitions between linear and circular segments. The DMC-3425 performs all the complex computations of linear and circular interpolation, freeing the host PC from this time intensive task.

The coordinated motion mode is similar to the linear interpolation mode. Any pair of two axes may be selected for coordinated motion consisting of linear and circular segments. Note that only one pair of axes can be specified for coordinated motion at any given time.

Specifying Vector Segments

The motion segments are described by two commands; VP for linear segments and CR for circular segments. Once a set of linear segments and/or circular segments have been specified, the sequence is ended with the command VE. This defines a sequence of commands for coordinated motion. Immediately prior to the execution of the first coordinated movement, the controller defines the current position to be zero for all movements in a sequence. Note: This 'local' definition of zero does not affect the absolute coordinate system or subsequent coordinated motion sequences.

The command, VP a,b specifies the coordinates of the end points of the vector movement with respect to the starting point. The command, CR r,θ,δ define a circular arc with a radius r, starting angle of θ, and a traversed angle δ. The notation for θ is that zero corresponds to the positive horizontal direction, and for both θ and δ, the counter-clockwise (CCW) rotation is positive.

Up to 511 segments of CR or VP may be specified in a single sequence and must be ended with the command VE. The motion can be initiated with a Begin Sequence (BGS) command. Once motion starts, additional segments may be added.

The Clear Sequence (CS) command can be used to remove previous VP and CR commands that were stored in the buffer prior to the start of the motion. To stop the motion, use the instructions STS or AB1. ST stops motion at the specified deceleration. AB1 aborts the motion instantaneously.

The Vector End (VE) command must be used to specify the end of the coordinated motion. This command tells the controller to decelerate to a stop following the last motion in the sequence. If a VE command is not given, an Abort (AB1) must be used to abort the coordinated motion sequence.

The user must keep enough motion segments in the DMC-3425 sequence buffer to ensure continuous motion. If the controller receives no additional motion segments and no VE command, the controller will stop motion instantly at the last vector. There will be no controlled deceleration. LM? or _LM returns the available spaces for motion segments that can be sent to the buffer. 511 returned means the buffer is empty and 511 segments can be sent. A zero means the buffer is full and no additional segments can be sent. As long as the buffer is not full, additional segments can be sent at the PCI bus speed.

The operand _CS can be used to determine the value of the segment counter.

Additional commands

The commands VS n, VA n and VD n are used for specifying the vector speed, acceleration, and deceleration.

VT is the motion smoothing constant used for coordinated motion.

Specifying Vector Speed for Each Segment:

The vector speed may be specified by the immediate command VS. It can also be attached to a motion segment with the instructions

VP x,y < n >m

CR r,θ,δ < n >m

The first parameter, <n, is equivalent to commanding VSn at the start of the given segment and will cause an acceleration toward the new commanded speeds, subjects to the other constraints.

The second parameter, > m, requires the vector speed to reach the value m at the end of the segment. Note that the function > m may start the deceleration within the given segment or during previous segments, as needed to meet the final speed requirement, under the given values of VA and VD.

Note, however, that the controller works with one > m command at a time. As a consequence, one function may be masked by another. For example, if the function >100000 is followed by >5000, and the distance for deceleration is not sufficient, the second condition will not be met. The controller will attempt to lower the speed to 5000, but will reach that at a different point.

Changing Feedrate:

The command VR n allows the feedrate, VS, to be scaled from 0 and 10 times with a resolution of .0001. This command takes effect immediately and causes VS scaled. VR also applies when the vector speed is specified with the '<' operator. This is a useful feature for feedrate override. VR does not ratio the accelerations. For example, VR .5 results in the specification VS 2000 act as VS 1000.

Compensating for Differences in Encoder Resolution:

By default, the DMC-3425 uses a scale factor of 1:1 for the encoder resolution when used in vector mode. If this is not the case, the command, ES can be used to scale the encoder counts. The ES command accepts two arguments that represent the ratio of the encoder resolutions. For more information refer to ES in the Command Reference.

Trippoints:

The AV n command is the After Vector trippoint, which waits for the vector relative distance of n to occur before executing the next command in a program.

Command Summary - Coordinated Motion Sequence

COMMAND	DESCRIPTION
VM m,n	Specifies the axes for the planar motion where m and n represent the planar axes.
VP m,n	Return coordinate of last point, where m=A,B,C or D.
CR r,θ,δ	Specifies arc segment where r is the radius, θ is the starting angle and δ is the travel angle. Positive direction is CCW.
VS n	Specify vector speed or feedrate of sequence.
VA n	Specify vector acceleration along the sequence.
VD n	Specify vector deceleration along the sequence.
VR n	Specify vector speed ratio
BGS	Begin motion sequence
CS	Clear sequence.
AV n	Trippoint for After Relative Vector distance, n.
AMS	Holds execution of next command until Motion Sequence is complete.
ES m,n	Ellipse scale factor.
VT	S curve smoothing constant for coordinated moves
LM?	Return number of available spaces for linear and circular segments in DMC-3425 sequence buffer. Zero means buffer is full. 512 means buffer is empty.

Operand Summary - Coordinated Motion Sequence

COMMAND	DESCRIPTION
_VPM	The absolute coordinate of the axes at the last intersection along the sequence.
_AV	Distance traveled.
_LM	Number of available spaces for linear and circular segments in DMC-3425 sequence buffer. Zero means buffer is full. 512 means buffer is empty.
_CS	Segment counter - Number of the segment in the sequence, starting at zero.
_VE	Vector length of coordinated move sequence.

When AV is used as an operand, _AV returns the distance traveled along the sequence.

The operands _VPA and _VPB can be used to return the coordinates of the last point specified along the path.

Example:

Traverse the path shown in Fig. 6.3. Feedrate is 20000 counts/sec. Plane of motion is AB

Instruction	**Interpretation**
VM AB	Specify motion plane
VS 20000	Specify vector speed
VA 1000000	Specify vector acceleration
VD 1000000	Specify vector deceleration
VP -4000,0	Segment AB
CR 1500,270,-180	Segment BC
VP 0,3000	Segment CD
CR 1500,90,-180	Segment DA
VE	End of sequence
BGS	Begin Sequence

The resulting motion starts at the point A and moves toward points B, C, D, A. Suppose that we interrogate the controller when the motion is halfway between the points A and B.

The value of _AV is 2000

The value of _CS is 0

_VPA and _VPB contain the absolute coordinate of the point A

Suppose that the interrogation is repeated at a point, halfway between the points C and D.

The value of _AV is 4000+1500π+2000=10,712

The value of _CS is 2

_VPA, _VPB contain the coordinates of the point C

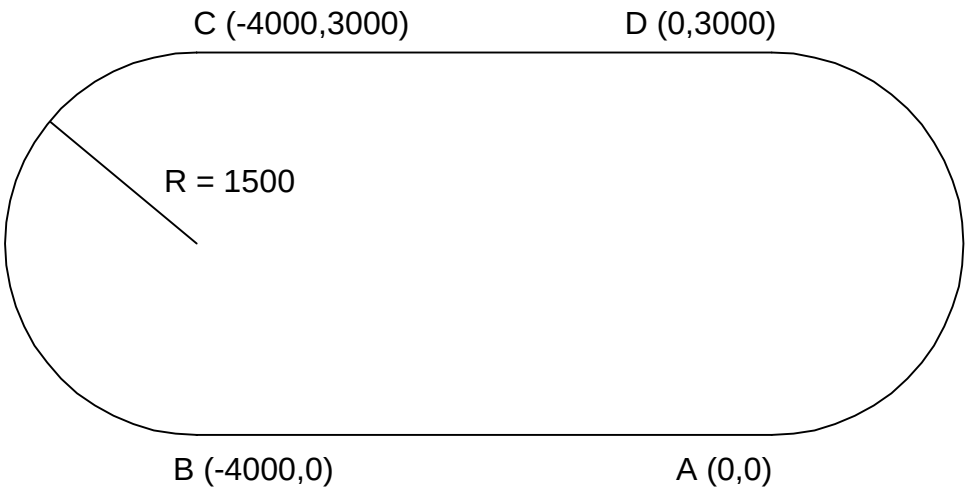


Figure 6.3 - The Required Path

Electronic Gearing (Local Mode)

This mode allows one axis to be electronically geared to the other axis. The master may rotate in both directions and the geared axes will follow at the specified gear ratio. The gear ratio may be different for each axis and changed during motion.

The command GA specifies the master axis. GR n,n specifies the gear ratios for the slaves where the ratio may be a number between +/-127.9999 with a fractional resolution of .0001. There are two modes: standard gearing and gantry mode. The gantry mode is enabled with the command GM. GR 0,0 turns off gearing in both modes. A limit switch or ST command disables gearing in the standard mode but not in the gantry mode.

The command GM n,n selects the axes to be controlled under the gantry mode. The parameter 1 enables gantry mode, and 0 disables it.

GR causes the specified axes to be geared to the actual position of the master. The master axis is commanded with motion commands such as PR, PA, or JG.

When the master axis is driven by the controller in the jog mode or an independent motion mode, it is possible to define the master as the command position of that axis, rather than the actual position. The designation of the commanded position master is by the letter C. For example, GACD indicates that the gearing is the commanded position of D.

Electronic gearing allows the geared motor to perform a second independent or coordinated move in addition to the gearing. For example, when a geared motor follows a master at a ratio of 1:1, it may be advanced an additional distance with PR, JG, VP, or LI commands.

Command Summary - Electronic Gearing

COMMAND	DESCRIPTION
GA n	Specifies master axes for gearing where: n = A,B for main encoder as master
	n = CA, CB for commanded position.
GR n,n	Sets gear ratio for slave axes. 0 disables electronic gearing for specified axis.
GM n,n	1 sets gantry mode, 0 disables gantry mode
MF n,n	Trippoint for forward motion past specified value. Only one field may be used.
MR n,n	Trippoint for reverse motion past specified value. Only one field may be used.
GA?	Retuns the GA command setting

Example – Electronic Gearing

Objective: Gear an A-axis slave motor at a speed of 2.5 times the speed of the B-axis master.

GAB	Specify B-axis as the master for A
GR2.5	Specify gear ratio for A to be 2.5 times the B axis master.

Example - Gantry Mode

In applications where both the master and the follower are controlled by the DMC-3425 controller, it may be desired to synchronize the follower with the commanded position of the master, rather than the actual position. This eliminates the possibility of an oscillation on the master passing the oscillation on to the slave.

For example, assume that a gantry is driven by two axes, A and B, one on each side. This requires the gantry mode for strong coupling between the motors. The A-axis is the master and the B-axis is the follower. To synchronize B with the commanded position of A, use the instructions:

GA, CA	Specify the commanded position of A as master for B.
GR,1	Set gear ratio for B as 1:1
GM,1	Set gantry mode
PR 3000	Command A motion
BG A	Start motion on A axis

You may also perform profiled position corrections in the electronic gearing mode. Suppose, for example, that you need to advance the slave 10 counts. Simply command

IP ,10	Specify an incremental position movement of 10 on the B axis.

Under these conditions, this IP command is equivalent to:

PR,10	Specify position relative movement of 10 on the B axis
BGB	Begin motion on the B axis

Often the correction is quite large. Such requirements are common when synchronizing cutting knives or conveyor belts.

Example - Synchronize two conveyor belts with trapezoidal velocity correction.

Instruction	**Interpretation**
GA,A	Define A as the master axis for B.
GR,2	Set gear ratio 2:1 for B
PR,300	Specify correction distance
SP,5000	Specify correction speed
AC,100000	Specify correction acceleration
DC,100000	Specify correction deceleration
BGB	Start correction

Electronic Cam (Local Mode)

The electronic cam is a motion control mode that enables the periodic synchronization of several axes of motion. Similar to the gearing mode, the DMC-3425 uses only A and B main axes as the master or slave.

The electronic cam is a more general type of electronic gearing which allows a table-based relationship between the axes. It allows synchronizing all the controller axes.

To illustrate the procedure of setting the cam mode, consider the cam relationship for the slave axis B, when the master is A. Such a graphic relationship is shown in Figure 6.4.

Step 1. Selecting the master axis

The first step in the electronic cam mode is to select the master axis. This is done with the instruction

EAp where p = A,B

p is the selected master axis

For the given example, since the master is a, we specify EAA

Step 2. Specify the master cycle and the change in the slave axis (es).

In the electronic cam mode, the position of the master is always expressed within one cycle. In this example, the position of a is always expressed in the range between 0 and 6000. Similarly, the slave position is also redefined such that it starts at zero and ends at 1500. At the end of a cycle when the master is 6000 and the slave is 1500, the positions of both a and b are redefined as zero. To specify the master cycle and the slave cycle change, we use the instruction EM.

```
EM a,b
```

where a,b specify the cycle of the master and the total change of the slaves over one cycle.

The cycle of the master is limited to 8,388,607 whereas the slave change per cycle is limited to 2,147,483,647. If the change is a negative number, the absolute value is specified. For the given example, the cycle of the master is 6000 counts and the change in the slave is 1500. Therefore, we use the instruction:

```
EM 6000,1500
```

Step 3. Specify the master interval and starting point.

Next we need to construct the ECAM table. The table is specified at uniform intervals of master positions. Up to 256 intervals are allowed. The size of the master interval and the starting point are specified by the instruction:

```
EP m,n
```

where m is the interval width in counts, and n is the starting point.

For the given example, we can specify the table by specifying the position at the master points of 0, 2000, 4000 and 6000. We can specify that by

```
EP 2000,0
```

Step 4. Specify the slave positions.

Next, we specify the slave positions with the instruction

```
ET[n]=x,y
```

where n indicates the order of the point.

The value, n, starts at zero and may go up to 256. The parameters x,y indicate the corresponding slave position. For this example, the table may be specified by

```
ET[0]=,0
ET[1]=,3000
ET[2]=,2250
ET[3]=,1500
```

This specifies the ECAM table.

Step 5. Enable the ECAM

To enable the ECAM mode, use the command

```
EB n
```

where n=1 enables ECAM mode and n=0 disables ECAM mode.

Step 6. Engage the slave motion

To engage the slave motion, use the instruction

```
EG a,b
```

where a,b are the master positions at which the corresponding slaves must be engaged.

If the value of any parameter is outside the range of one cycle, the cam engages immediately. When the cam is engaged, the slave position is redefined, modulo one cycle.

Step 7. Disengage the slave motion

To disengage the cam, use the command

```
EQ a,b
```

where a,b are the master positions at which the corresponding slave axes are disengaged.

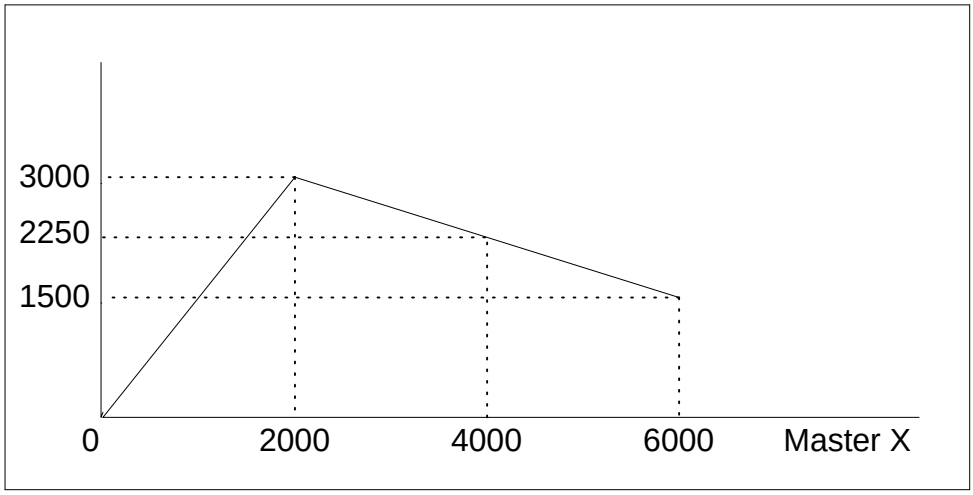


Figure 6.4 - Electronic Cam Example

This disengages the slave axis at a specified master position. If the parameter is outside the master cycle, the stopping is instantaneous.

To illustrate the complete process, consider the cam relationship described by

the equation:

$$Y = 0.5 * X + 100 \sin (0.18*X)$$

where A is the master, with a cycle of 2000 counts.

The cam table can be constructed manually, point by point, or automatically by a program. The following program includes the set-up.

The instruction EAA defines A as the master axis. The cycle of the master is

2000. Over that cycle, B varies by 1000. This leads to the instruction EM 2000,1000.

Suppose we want to define a table with 100 segments. This implies increments of 20 counts each. If the master points are to start at zero, the required instruction is EP 20,0.

The following routine computes the table points. As the phase equals 0.18X and A varies in increments of 20, the phase varies by increments of 3.6°. The program then computes the values of B according to the equation and assigns the values to the table with the instruction ET[N] = ,B.

Instruction	Interpretation
#SETUP	Label
EAA	Select A as master
EM 2000,1000	Cam cycles
EP 20,0	Master position increments
N = 0	Index
#LOOP	Loop to construct table from equation
P = N∗3.6	Note 3.6 = 0.18∗20
S = @SIN [P] ∗100	Define sine position
Y = N ∗10+S	Define slave position
ET [N] =, B	Define table
N = N+1	
JP #LOOP, N<=100	Repeat the process
EN	

Now suppose that the slave axis is engaged with a start signal, input 1, but that both the engagement and disengagement points must be done at the center of the cycle: A = 1000 and B = 500. This implies that B must be driven to that point to avoid a jump.

This is done with the program:

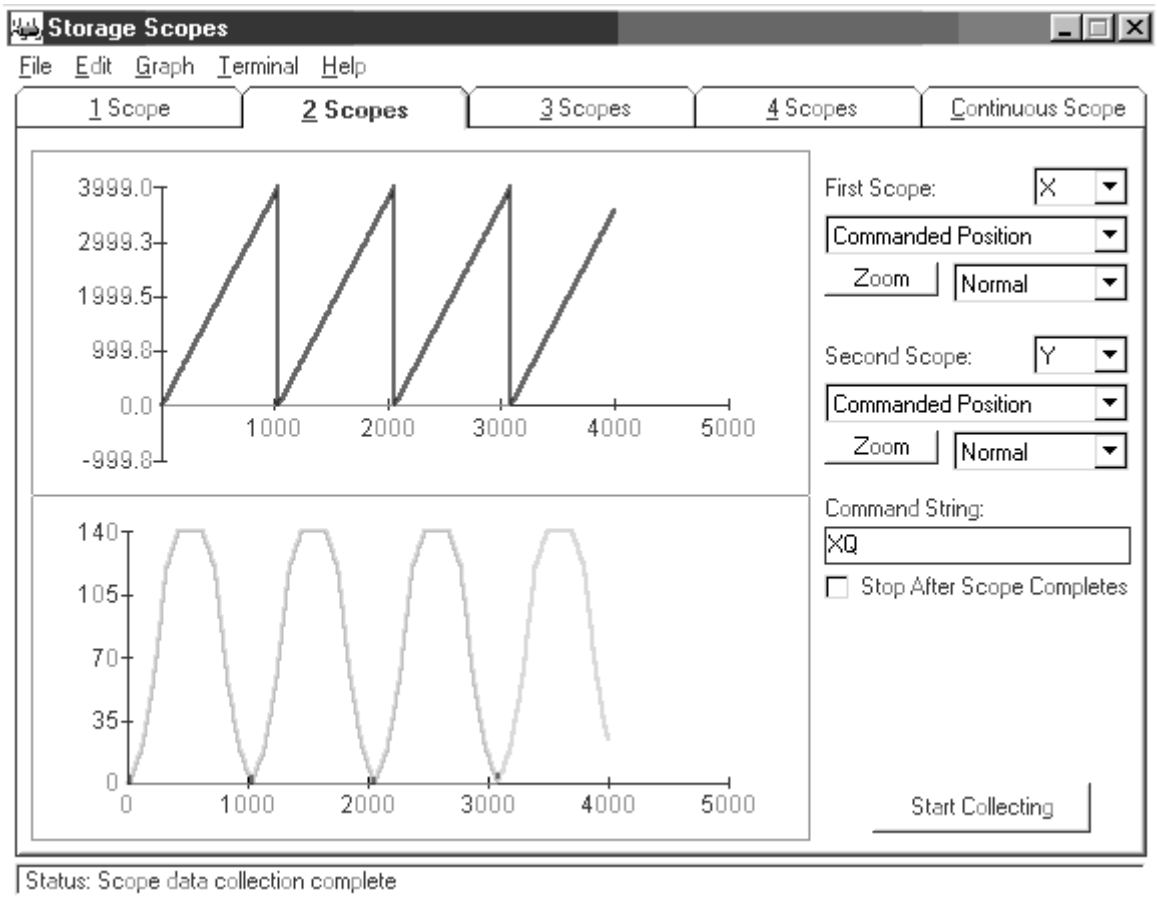
Instruction	Interpretation
#RUN	Label
EB1	Enable cam
PA,500	B starting position
SP,5000	B speed
BGB	Move B motor
AM	After B moved
AI1	Wait for start signal
EG,1000	Engage slave
AI – 1	Wait for stop signal
EQ,1000	Disengage slave
EN	End

The following example illustrates a cam program with a master axis, A, and a single slave B.

Instruction	Interpretation
#A;V1=0	Label; Initialize variable
PA 0,0;BGAB;AMAB	Go to position 0,0 on A and B axes
EA A	A axis as the Master for ECAM
EM 4000,0	Change for A is 4000, zero for B
EP400,0	ECAM interval is 400 counts with zero start
ET[0]=,0	When master is at 0 position; 1st point.
ET[1]=,20	2nd point in the ECAM table

ET[2]=,60	3rd point in the ECAM table
ET[3]=,120	4th point in the ECAM table
ET[4]=,140	5th point in the ECAM table
ET[5]=,140	6th point in the ECAM table
ET[6]=,140	7th point in the ECAM table
ET[7]=,120	8th point in the ECAM table
ET[8]=,60	9th point in the ECAM table
ET[9]=,20	10th point in the ECAM table
ET[10]=,0	Starting point for next cycle
EB 1	Enable ECAM mode
JGA=4000	Set A to jog at 4000
EG ,0	Engage both A and B when Master = 0
BGA	Begin jog on A axis
#LOOP;JP#LOOP,V1=0	Loop until the variable is set
EQ,2000	Disengage B when Master = 2000
MF2000	Wait until the Master goes to 2000
ST A	Stop the A axis motion
EB 0	Exit the ECAM mode
EN	End of the program

The above example shows how the ECAM program is structured and how the commands can be given to the controller. The next page provides the results captured by the WSDK program. This shows how the motion will be seen during the ECAM cycles. The first graph is for the A axis, the master, and the second graph shows the cycle on the B axis.



Contour Mode (Local Mode)

The DMC-3425 also provides a contouring mode. This mode allows any arbitrary position curve to be prescribed for any motion axes. This is ideal for following computer generated paths such as parabolic, spherical or user-defined profiles. The path is not limited to straight line and arc segments and the path length may be infinite.

Specifying Contour Segments

The Contour Mode is specified with the command, CM. For example, CMAB specifies contouring on the A and B axes. Any axes that are not being used in the contouring mode may be operated in other modes.

A contour is described by position increments which are described with the command, CD a,b over a time interval, DT n. The parameter, n, specifies the time interval. The time interval is defined as 2^n ms, where n is a number between 1 and 8. The controller performs linear interpolation between the specified increments, where one point is generated for each millisecond.

Consider, for example, the trajectory shown in Fig. 6.5. The position A may be described by the points:

Point 1	A=0 at T=0ms
Point 2	A=48 at T=4ms
Point 3	A=288 at T=12ms
Point 4	A=336 at T=28ms

The same trajectory may be represented by the increments

Increment 1	DA=48	Time Increment =4	DT=2
Increment 2	DA=240	Time Increment =8	DT=3
Increment 3	DA=48	Time Increment =16	DT=4

When the controller receives the command to generate a trajectory along these points, it interpolates linearly between the points. The resulting interpolated points include the position 12 at 1 msec, position 24 at 2 msec, etc.

The programmed commands to specify the above example are:

Instruction	**Description**
#A	Label
CMA	Specifies A axis for contour mode
DT 2	Specifies first time interval, 2^2 ms
CD 48;WC	Specifies first position increment
DT 3	Specifies second time interval, 2^3 ms
CD 240;WC	Specifies second position increment
DT 4	Specifies the third time interval, 2^4 ms
CD 48;WC	Specifies the third position increment
DT0;CD0	Exits contour mode
EN	

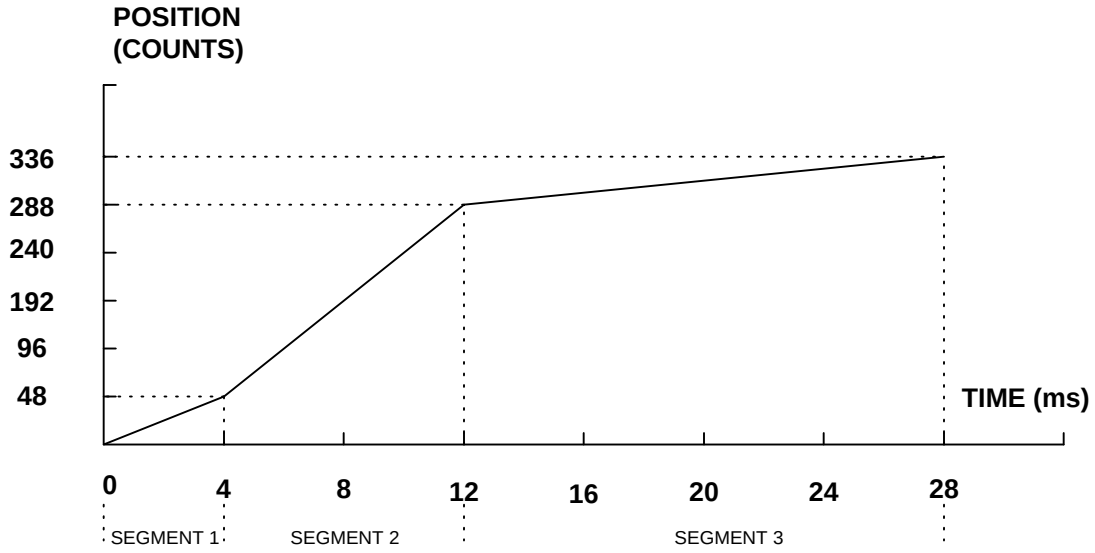


Figure 6.5 - The Required Trajectory

Additional Commands

The command, WC, is used as a trippoint "When Complete" or “Wait for Contour Data”. This allows the DMC-3425 to use the next increment only when it is finished with the previous one. Zero parameters for DT followed by zero parameters for CD exit the contour mode.

If no new data record is found and the controller is still in the contour mode, the controller waits for new data. No new motion commands are generated while waiting. If bad data is received, the controller responds with a ?.

Command Summary - Contour Mode

Command	Description
CM AB	Specifies which axes for contouring mode. Any non-contouring axes may be operated in other modes.
CD a,b	Specifies position increment over time interval. Range is +/-32,000. Zero ends contour mode.
DT n	Specifies time interval 2^n msec for position increment, where n is an integer between 1 and 8. Zero ends contour mode. If n does not change, it does not need to be specified with each CD.
WC	Waits for previous time interval to be complete before next data record is processed.

Operand Summary - Contour Mode

Operand	Description
_CS	Return segment number

General Velocity Profiles

The Contour Mode is ideal for generating an arbitrary velocity profile. The velocity profile can be specified as a mathematical function or as a collection of points.

The design includes two parts: Generating an array with data points and running the program.

Generating an Array - An Example

Consider the velocity and position profiles shown in Fig. 6.6. The objective is to rotate a motor a distance of 6000 counts in 120 ms. The velocity profile is sinusoidal to reduce the jerk and the system vibration. If we describe the position displacement in terms of A counts in B milliseconds, we can describe the motion in the following manner:

$$\boldsymbol{\omega} = (A/B)\,[1 - \cos(2\pi T/B)]$$

$$\mathbf{X} = (AT/B) - (A/2\pi)\sin(2\pi T/B)$$

Note: $\boldsymbol{\omega}$ is the angular velocity; **X** is the position; and T is the variable, time, in milliseconds.

In the given example, A=6000 and B=120, the position and velocity profiles are:

$$\mathbf{X} = 50T - (6000/2\pi)\sin(2\pi\, T/120)$$

Note that the velocity, ω, in count/ms, is

$\omega = 50\ [1 - \cos 2\pi\ T/120]$

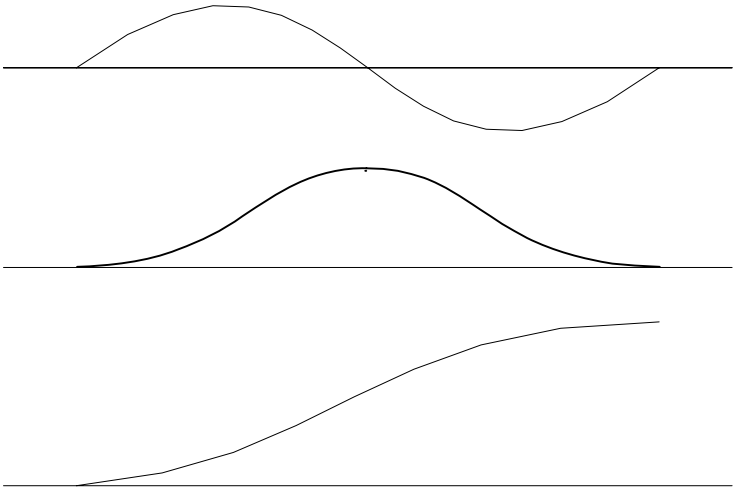


Figure 6.6 - Velocity Profile with Sinusoidal Acceleration

The DMC-3425 can compute trigonometric functions. However, the argument must be expressed in degrees. Using our example, the equation for X is written as:

$X = 50T - 955 \sin 3T$

A complete program to generate the contour movement in this example is given below. To generate an array, we compute the position value at intervals of 8 ms. This is stored at the array POS. Then, the difference between the positions is computed and is stored in the array DIF. Finally the motors are run in the contour mode.

Contour Mode Example

Instruction	**Interpretation**
#POINTS	Program defines A points
DM POS[16]	Allocate memory
DM DIF[15]	
C=0	Set initial conditions, C is index
T=0	T is time in ms
#A	
V1=50*T	
V2=3*T	Argument in degrees
V3=-955*@SIN[V2]+V1	Compute position
V4=@INT[V3]	Integer value of V3
POS[C]=V4	Store in array POS
T=T+8	
C=C+1	
JP #A,C<16	
#B	Program to find position differences

```
C=0
#C
D=C+1
DIF[C]=POS[D]-POS[C]      Compute the difference and store
C=C+1
JP #C,C<15
EN                        End first program
#RUN                      Program to run motor
CMA                       Contour Mode
DT3                       4 millisecond intervals
C=0
#E
CD DIF[C]                 Contour Distance is in DIF
WC                        Wait for completion
C=C+1
JP #E,C<15
DT0
CD0                       Stop Contour
EN                        End the program
```

Teach (Record and Play-Back)

Several applications require teaching the machine a motion trajectory. Teaching can be accomplished using the DMC-3425 automatic array capture feature to capture position data. The captured data may then be played back in the contour mode. The following array commands are used:

DM C[n]	Dimension array
RA C[]	Specify array for automatic record
RD _TPA	Specify data for capturing (such as _TPA or _TPB)
RC n,m	Specify capture time interval where n is 2n msec, m is number of records to be captured
RC? or _RC	Returns a 1 if recording

Record and Playback Example:

Instruction	**Interpretation**
#RECORD	Begin Program
DM APOS[501]	Dimension array with 501 elements
RA APOS[]	Specify automatic record
RD _TPA	Specify A position to be captured
MOA	Turn A motor off
RC2	Begin recording; 4 msec interval
#A;JP#A,_RC=1	Continue until done recording
#COMPUTE	Compute DX
DM DX[500]	Dimension Array for DX
C=0	Initialize counter

#L	Label
D=C+1	
DELTA=XPOS[D]-XPOS[C]	Compute the difference
DX[C]=DELTA	Store difference in array
C=C+1	Increment index
JP #L,C<500	Repeat until done
#PLAYBCK	Begin Playback
CMA	Specify contour mode
DT2	Specify time increment
I=0	Initialize array counter
#B	Loop counter
CD XPOS[I];WC	Specify contour data I=I+1 Increment array counter JP #B,I<500 Loop until done
DT 0;CD0	End contour mode
EN	End program

For additional information about automatic array capture, see Chapter 7, Arrays.

Virtual Axis (Local Mode)

The DMC-3425 controller has an internal motion profiler, also referred to as a virtual axis. This axis is designated as the N axis and has no encoder input and no DAC output. With the N axis, a commanded position profile can be generated using the following modes of motion:

Mode of Motion	Virtual Axis usage	Commands
Relative Independent Axis Positioning	N axis profile is specified with a relative distance, velocity, acceleration and deceleration. The N axis profile follows the prescribed velocity profile.	PRN=<value> ACN=<value> DCN=<value> SPN=<value>
Absolute Independent Axis Positioning	N axis profile is specified with an absolute distance, velocity, acceleration and deceleration. The N axis profile follows the prescribed velocity profile.	PAN=<value> ACN=<value> DCN=<value> SPN=<value>
Independent Jogging	N axis profile is specified with a prescribed velocity with no final endpoint. The motion is specified with velocity, acceleration and deceleration. Motion stops on Stop command.	JGN=<value> ACN=<value> DCN=<value> STN=<value>
Vector Motion	N axis profile replaces one of the 2 axes specified for 2-D motion. Vector velocity, vector acceleration and vector deceleration are specified. The vector motion follows the prescribed velocity profile.	VMxN VMNx x represents the 2nd axis used for vector motion
Electronic Gearing	N axis can be used as a master axis for gearing	GAx=N GA N,N
Electronic Cam	N axis can be used as a master axis for electronic CAM	EA N EMN=<value>

The main use of the virtual axis is to serve as a virtual master in ECAM modes, and to perform an unnecessary part of a vector mode. These applications are illustrated by the following examples.

Ecam Master Example

Suppose that the motion of the AB axes is constrained along a path that can be described by an electronic cam table. Further assume that the ecam master is not an external encoder but has to be a controlled variable.

This can be achieved by defining the N axis as the master with the command EAN and setting the modulo of the master with a command such as EMN= 4000. Next, the table is constructed. To move the constrained axes, simply command the N axis in the jog mode or with the PR and PA commands.

For example,

```
PAN = 2000
BGN
```

will cause the AB axes to move to the corresponding points on the motion cycle.

Sinusoidal Motion Example

The A axis must perform a sinusoidal motion of 10 cycles with an amplitude of 1000 counts and a frequency of 20 Hz.

This can be performed by commanding the A and N axes to perform circular motion. Note that the value of VS must be

$$VS = 2\pi * R * F$$

where R is the radius, or amplitude and F is the frequency in Hz.

Set VA and VD to maximum values for the fastest acceleration.

Instruction	Interpretation
VMAN	Select Axes
VA 68000000	Maximum Acceleration
VD 68000000	Maximum Deceleration
VS 125664	VS for 20 Hz
CR 1000, -90, 3600	Ten Cycles
VE	
BGS	

Stepper Motor Operation

When configured for stepper motor operation, several commands are interpreted differently than from servo mode. The following describes operation with stepper motors.

NOTE: If two steppers are to be used with the DMC-3425, the controller must be ordered from the factory as a DMC-3425-Stepper.

Specifying Stepper Motor Operation

In order to command stepper motor operation, the appropriate stepper mode jumpers must be installed. See chapter 2 for this installation.

Stepper motor operation is specified by the command MT. The argument for MT is as follows:

2	specifies a stepper motor with active low step output pulses
-2	specifies a stepper motor with active high step output pulses
2.5	specifies a stepper motor with active low step output pulses and reversed direction
-2.5	specifies a stepper motor with active high step output pulse and reversed direction

Stepper Motor Smoothing

The command, KS, provides stepper motor smoothing. The effect of the smoothing can be thought of as a simple Resistor-Capacitor (single pole) filter. The filter occurs after the motion profiler and has the effect of smoothing out the spacing of pulses for a more smooth operation of the stepper motor. Use of KS is most applicable when operating in full step or half step operation. KS will cause the step pulses to be delayed in accordance with the time constant specified.

When operating with stepper motors, you will always have some amount of stepper motor smoothing, KS. Since this filtering effect occurs after the profiler, the profiler may be ready for additional moves before all of the step pulses have gone through the filter. It is important to consider this effect since steps may be lost if the controller is commanded to generate an additional move before the previous move has been completed. See the discussion below, *Monitoring Generated Pulses vs. Commanded Pulses*.

The general motion smoothing command, IT, can also be used. The purpose of the command, IT, is to smooth out the motion profile and decrease 'jerk' due to acceleration.

Monitoring Generated Pulses vs. Commanded Pulses

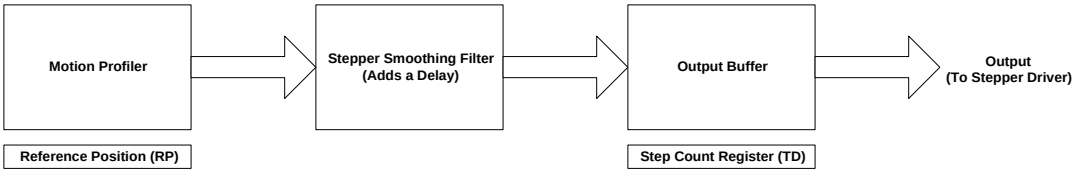
For proper controller operation, it is necessary to make sure that the controller has completed generating all step pulses before making additional moves. This is most particularly important if you are moving back and forth. For example, when operating with servo motors, the trippoint AM (After Motion) is used to determine when the motion profiler is complete and is prepared to execute a new motion command. However when operating in stepper mode, the controller may still be generating step pulses when the motion profiler is complete. This is caused by the stepper motor smoothing filter, KS. To understand this, consider the steps the controller executes to generate step pulses:

First, the controller generates a motion profile in accordance with the motion commands.

Second, the profiler generates pulses as prescribed by the motion profile. The pulses that are generated by the motion profiler can be monitored by the command, RP (Reference Position). RP gives the absolute value of the position as determined by the motion profiler. The command, DP, can be used to set the value of the reference position. For example, DP 0, defines the reference position of the A axis to be zero.

Third, the output of the motion profiler is filtered by the stepper smoothing filter. This filter adds a delay in the output of the stepper motor pulses. The amount of delay depends on the parameter that is specified by the command, KS. As mentioned earlier, there will always be some amount of stepper motor smoothing. The default value for KS is 2, which corresponds to a time constant of 6 sample periods.

Fourth, the output of the stepper smoothing filter is buffered and is available for input to the stepper motor driver. The pulses that are generated by the smoothing filter can be monitored by the command, TD (Tell Dual). TD gives the absolute value of the position as determined by actual output of the buffer. The command, DP sets the value of the step count register as well as the value of the reference position. For example, DP 0, defines the reference position of the A axis to be zero.



Motion Complete Trippoint

When used in stepper mode, the MC command will hold up execution of the proceeding commands until the controller has generated the same number of steps out of the step count register as specified in the commanded position. The MC trippoint (Motion Complete) is generally more useful than AM trippoint (After Motion) since the step pulses can be delayed from the commanded position due to stepper motor smoothing.

Using an Encoder with Stepper Motors

An encoder may be used on a stepper motor to check the actual motor position with the commanded position. If an encoder is used, it must be connected to the main encoder input.

NOTE: The auxiliary encoder is not available while operating with stepper motors. The position of the encoder can be interrogated by using the command, TP. The position value can be defined by using the command, DE.

NOTE: Closed loop operation with a stepper motor is not possible.

Command Summary - Stepper Motor Operation

Command	Description
DE	Define Encoder Position (When using an encoder)
DP	Define Reference Position and Step Count Register
IT	Motion Profile Smoothing - Independent Time Constant
KS	Stepper Motor Smoothing
MT	Motor Type (2,-2,2.5 or -2.5 for stepper motors)
RP	Report Commanded Position
TD	Report number of step pulses generated by controller
TP	Tell Position of Encoder

Operand Summary - Stepper Motor Operation

Operand	Description
_DEa	Contains the value of the step count register for the ‘a’ axis
_DPa	Contains the value of the main encoder for the ‘a’ axis
_ITa	Contains the value of the Independent Time constant for the 'a' axis
_KSa	Contains the value of the Stepper Motor Smoothing Constant for the 'a' axis
_MTa	Contains the motor type value for the 'a' axis
_RPa	Contains the commanded position generated by the profiler for the ‘a’ axis
_TDa	Contains the value of the step count register for the ‘a’ axis
_TPa	Contains the value of the main encoder for the ‘a’ axis

Dual Loop (Auxiliary Encoder)

The DMC-3415 provides an interface for a second encoder except when configured for stepper motor operation or circular compare. Please note, the DMC-3425 has only a single encoder per axis. When used, the second encoder is typically mounted on the motor or the load, but may be mounted in any position. The most common use for the second encoder is backlash compensation, described below.

The second encoder may be a standard quadrature type, or it may provide pulse and direction. The controller also offers the provision for inverting the direction of the encoder rotation. The main and the auxiliary encoders are configured with the CE command. The command form is CEa, where the parameter a equals the sum of two integers m and n. m configures the main encoder and n configures the auxiliary encoder.

NOTE: This operation is not available when the DMC-3415 is configured for a stepper motor.

Using the CE Command

m=	Main Encoder	n=	Second Encoder
0	Normal quadrature	0	Normal quadrature
1	Pulse & direction	4	Pulse & direction
2	Reverse quadrature	8	Reversed quadrature
3	Reverse pulse & direction	12	Reversed pulse & direction

For example, to configure the main encoder for reversed quadrature, m=2, and a second encoder of pulse and direction, n=4, the total is 6, and the command for the A axis is

```
CE 6
```

Additional Commands for the Auxiliary Encoder

The command, DEa can be used to define the position of the auxiliary encoder. For example,

```
DE 500
```

sets the initial value.

The position of the auxiliary encoder may be interrogated with the command, DE?.

The auxiliary encoder position may be assigned to variables with the instructions

```
V1= _DEA
```

The command, TD a,b,c,d, returns the current position of the auxiliary encoder.

The command, DV a,b,c,d, configures the auxiliary encoder to be used for backlash compensation.

Backlash Compensation

There are two methods for backlash compensation using the auxiliary encoder:

1. Continuous dual loop
2. Sampled dual loop

To illustrate the problem, consider a situation in which the coupling between the motor and the load has a backlash. To compensate for the backlash, position encoders are mounted on both the motor and the load.

The continuous dual loop combines the two feedback signals to achieve stability. This method requires careful system tuning, and depends on the magnitude of the backlash. However, once successful, this method compensates for the backlash continuously.

The second method, the sampled dual loop, reads the load encoder only at the end point and performs a correction. This method is independent of the size of the backlash. However, it is effective only in point-to-point motion systems that require position accuracy only at the endpoint.

Example

Continuous Dual Loop

Connect the load encoder to the main encoder port and connect the motor encoder to the dual encoder port. The dual loop method splits the filter function between the two encoders. It applies the KP (proportional) and KI (integral) terms to the position error, based on the load encoder, and applies the KD (derivative) term to the motor encoder. This method results in a stable system.

The dual loop method is activated with the instruction DV (Dual Velocity), where

DV1

activates the dual loop and

DV0

disables the dual loop.

Note that the dual loop compensation depends on the backlash magnitude, and in extreme cases will not stabilize the loop. The proposed compensation procedure is to start with KP=0, KI=0 and to maximize the value of KD under the condition DV1. Once KD is found, increase KP gradually to a maximum value, and finally, increase KI, if necessary.

Sampled Dual Loop

In this example, we consider a linear slide that is run by a rotary motor via a lead screw. Since the lead screw has a backlash, it is necessary to use a linear encoder to monitor the position of the slide. For stability reasons, it is best to use a rotary encoder on the motor.

Connect the rotary encoder to the A-axis and connect the linear encoder to the auxiliary encoder of A. Assume that the required motion distance is one inch, and that this corresponds to 40,000 counts of the rotary encoder and 10,000 counts of the linear encoder.

The design approach is to drive the motor a distance, which corresponds to 40,000 rotary counts. Once the motion is complete, the controller monitors the position of the linear encoder and performs position corrections.

This is done by the following program.

Instruction	**Interpretation**
#DUALOOP	Label
CE 0	Configure encoder
DE0	Set initial value
PR 40000	Main move
BGA	Start motion
#CORRECT	Correction loop
AMA	Wait for motion completion
v1=10000-_DEA	Find linear encoder error
v2=-_TEA/4+v1	Compensate for motor error
JP#END,@ABS[v_2]<2	Exit if error is small

PR v_2*4	Correction move
BGA	Start correction
JP#CORRECT	Repeat
#END	
EN	

Motion Smoothing

The DMC-3425 controller allows the smoothing of the velocity profile to reduce the mechanical vibration of the system.

Trapezoidal velocity profiles have acceleration rates that change abruptly from zero to maximum value. The discontinuous acceleration results in jerk which causes vibration. The smoothing of the acceleration profile leads to a continuous acceleration profile and reduces the mechanical shock and vibration.

Using the IT and VT Commands:

When operating with servo motors, motion smoothing can be accomplished with the IT and VT command. These commands filter the acceleration and deceleration functions to produce a smooth velocity profile. The resulting velocity profile has continuous acceleration and results in reduced mechanical vibrations.

The smoothing function is specified by the following commands:

IT a	Independent time constant
VT n	Vector time constant

The command, IT, is used for smoothing independent moves of the type JG, PR, PA and the command, VT, is used to smooth vector moves of the type VM and LM.

The smoothing parameter a and n are numbers between 0 and 1 and determine the degree of filtering. The maximum value of 1 implies no filtering, resulting in trapezoidal velocity profiles. Smaller values of the smoothing parameters imply heavier filtering and smoother moves.

The following example illustrates the effect of smoothing. Fig. 6.7 shows the trapezoidal velocity profile and the modified acceleration and velocity.

Note that the smoothing process results in longer motion time.

Example

Instruction	**Interpretation**
PR 20000	Position
AC 100000	Acceleration
DC 100000	Deceleration
SP 5000	Speed
IT .5	Filter for smoothing
BG A	Begin

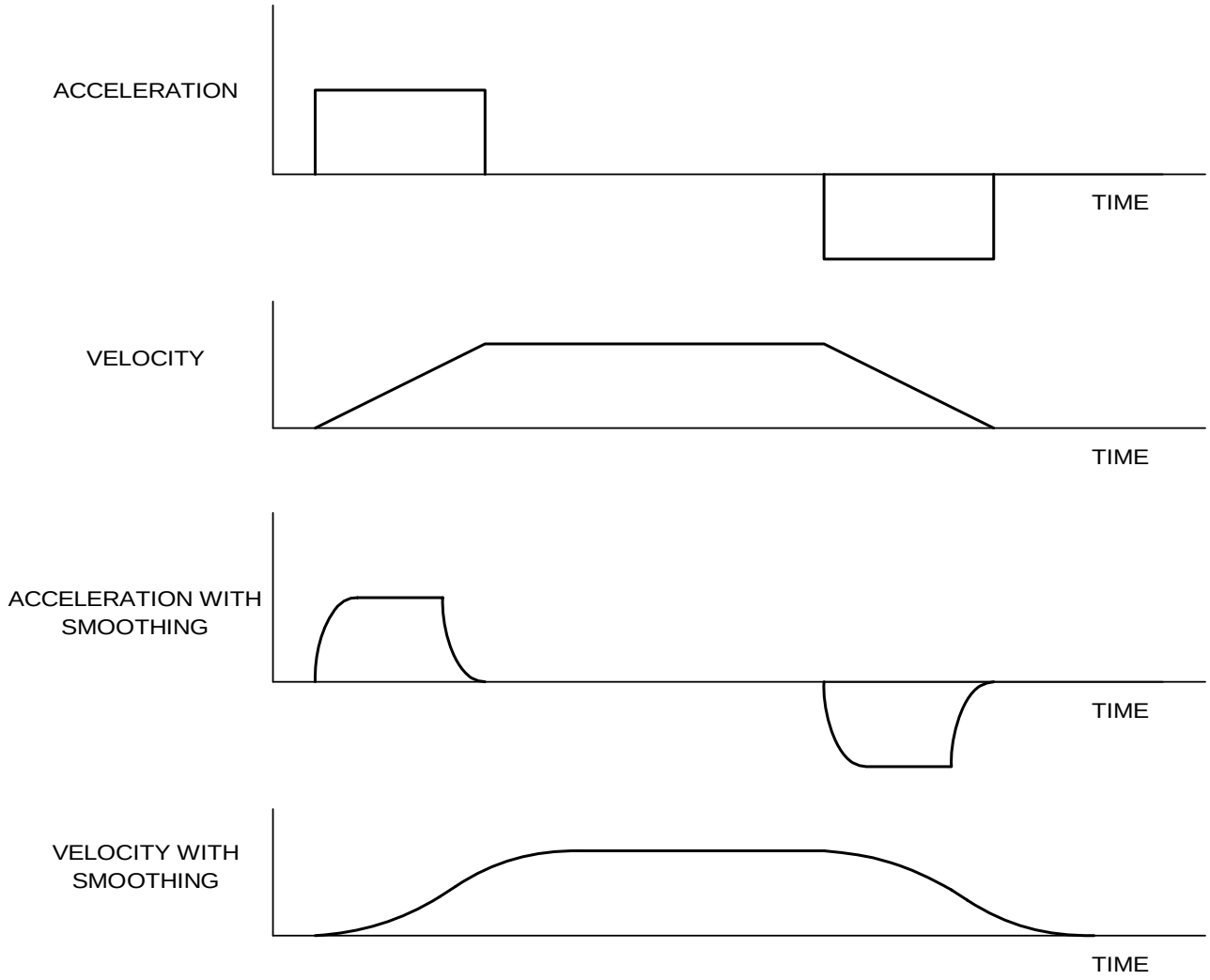


Figure 6.7 - Trapezoidal velocity and smooth velocity profiles

Homing

The Find Edge (FE) and Home (HM) instructions may be used to home the motor to a mechanical reference. This reference is connected to the Home input line. The HM command initializes the motor to the encoder index pulse in addition to the Home input. The configure command (CN) is used to define the polarity of the home input.

The Find Edge (FE) instruction is useful for initializing the motor to a home switch. The home switch is connected to the Homing Input. When the Find Edge command and Begin is used, the motor will accelerate up to the slew speed and slew until a transition is detected on the Homing line. The motor will then decelerate to a stop. A high deceleration value must be input before the find edge command is issued for the motor to decelerate rapidly after sensing the home switch. The velocity profile generated is shown in Fig. 6.7.

The Home (HM) command can be used to position the motor on the index pulse after the home switch is detected. This allows for finer positioning on initialization. The command sequence HM and BG causes the following sequence of events to occur.

1. Upon Begin, motor accelerates to the slew speed. The direction of its motion is determined by the state of the homing input. A zero (GND) will cause the motor to start

in the forward direction; +5V will cause it to start in the reverse direction. The CN command is used to define the polarity of the home input.

2. Upon detecting the home switch changing state, the motor begins decelerating to a stop.
3. The motor then traverses very slowly back until the home switch toggles again.
4. The motor then traverses forward until the encoder index pulse is detected.
5. The DMC-3425 defines the home position (0) as the position at which the index was detected.

Example

Instruction	Interpretation
#HOME	Label
AC 1000000	Acceleration Rate
DC 1000000	Deceleration Rate
SP 5000	Speed for Home Search
HM A	Home A
BG A	Begin Motion
AM A	After Complete
MG "AT HOME"	Send Message
EN	End
#EDGE	Label
AC 2000000	Acceleration rate
DC 2000000	Deceleration rate
SP 8000	Speed
FE B	Find edge command
BG B	Begin motion
AM B	After complete
MG "FOUND HOME"	Send message
DP,0	Define position as 0
EN	End

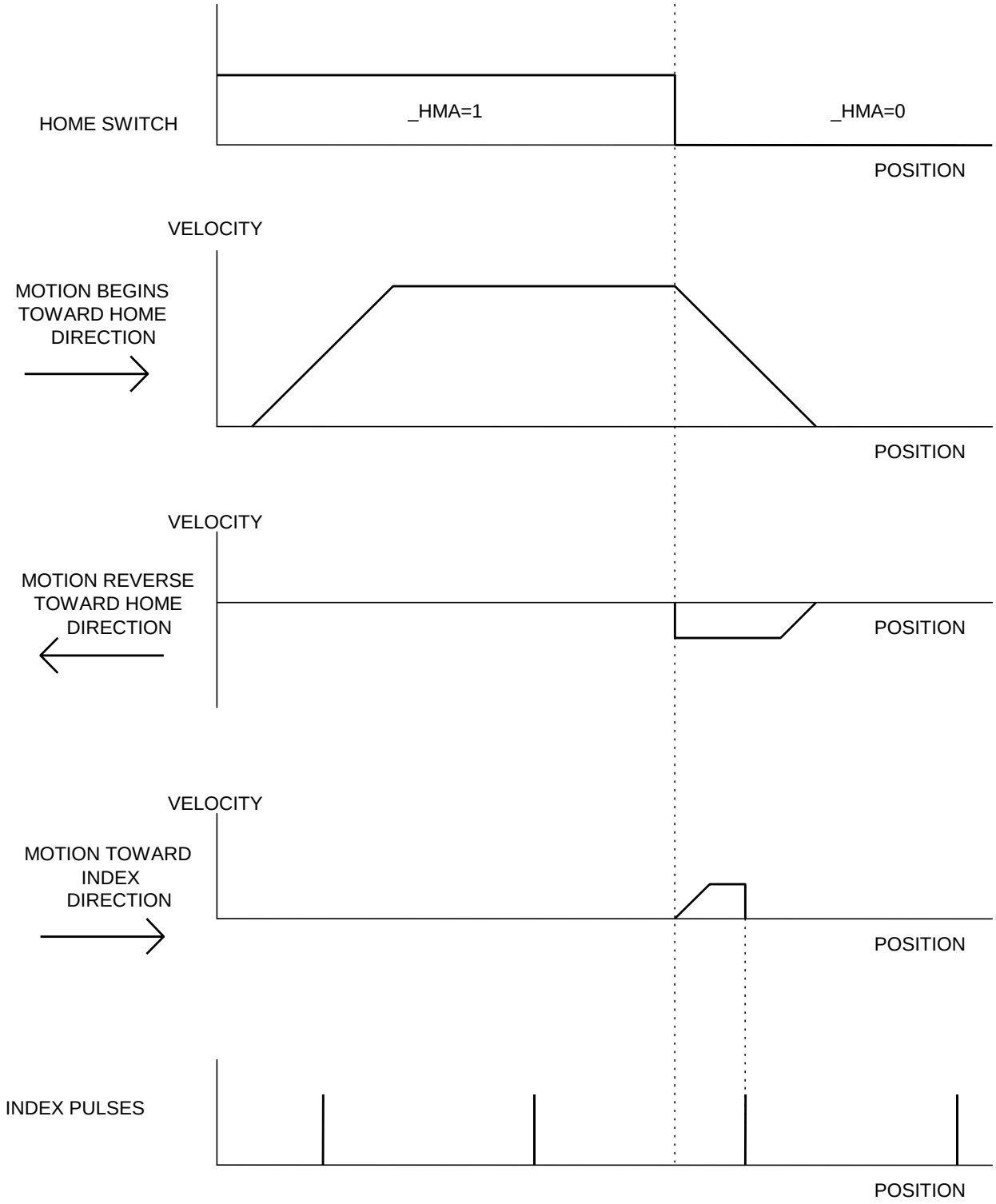


Figure 6.8 - Motion intervals in the Home sequence

Command Summary - Homing Operation

COMMAND	DESCRIPTION
FE ABCD	Find Edge Routine. This routine monitors the Home Input
FI ABCD	Find Index Routine - This routine monitors the Index Input
HM ABCD	Home Routine - This routine combines FE and FI as Described Above
SC ABCD	Stop Code
TS ABCD	Tell Status of Switches and Inputs

Operand Summary - Homing Operation

OPERAND	DESCRIPTION
_HMn	Contains the value of the state of the Home Input
_SCn	Contains stop code
_TSn	Contains status of switches and inputs

High Speed Position Capture (Latch)

Often it is desirable to capture the position precisely for registration applications. The DMC-3425 provides a position latch feature. This feature allows the position of the encoders of A or B axis to be captured when the latch input changes state. This function can be setup such that the position is captured when the latch input goes high or low. The inputs on these controllers are TTL. Latch time latency on a high or low going signal is less than 1μsec. Each axis has one general input associated to the axis for position capture:

Input	Function
IN1	A Axis Latch
IN2	B Axis Latch

The DMC-3425 software commands, AL and RL, are used to arm the latch and report the latched position. The steps to use the latch are as follows:

1. Give the AL AB command to arm the latch.
2. Test to see if the latch has occurred (Input goes low) by testing the operand, _ALA or _ALB. Example, V1=_ALA returns the state of the A latch into V1. V1 is 1 if the latch has not occurred.
3. After the latch has occurred, read the captured position with the command RL AB or RL AB command or monitor the value of the operands _RLA and _RLB.

NOTE: The latch must be re-armed after each latching event.

Example

Instruction	Interpretation
#Latch	Latch program
JG,5000	Jog B
BG B	Begin motion on B axis
AL B	Arm Latch for B axis
#Wait	#Wait label for loop
JP #Wait,_ALB=1	Jump to #Wait label if latch has not occurred
Result=_RLB	Set 'Result' equal to the reported position of B axis
Result=	Print result
EN	End

THIS PAGE LEFT BLANK INTENTIONALLY

Chapter 7 Application Programming

Overview

The DMC-3425 provides a powerful programming language that allows users to customize the controller for their particular application. Programs can be downloaded into the DMC-3425 memory freeing the host computer for other tasks. However, the host computer can send commands to the controller at any time, even while a program is being executed. Only ASCII commands can be used for application programming.

In addition to standard motion commands, the DMC-3425 provides commands that allow the DMC-3425 to make its own decisions. These commands include conditional jumps, event triggers and subroutines. For example, the command JP#LOOP, n<10 causes a jump to the label #LOOP if the variable n is less than 10.

For greater programming flexibility, the DMC-3425 provides user-defined variables, arrays and arithmetic functions. For example, with a cut-to-length operation, the length can be specified as a variable in a program that the operator can change as necessary.

Global vs. Local Programming

As mentioned previously, multiple DMC-3425 controllers can be connected through an Ethernet hub. The DMC-3425 controllers can be setup to operate in 2 modes; **LOCAL OPERATION** and **GLOBAL OPERATION**.

In Local Operation, the host computer can download a program to any DMC-3425 and all program commands refer to the two axes on the controller as A and B. Each controller operates independently. This type of program is referred to as a **LOCAL PROGRAM**.

In Global Operation, up to eight axes of DMC-3425 and DMC-3415 controllers act as a "virtual multi-axis controller". One DMC-3425 is designated as the master controller and the other controllers are designated as slave controllers. The host computer can download a program to the master DMC-3425. The master controller program contains commands that address all axes in the system. This **GLOBAL PROGAM** will operate as if it was a program on a traditional multi-axis controller. In addition, each slave controller can also be programmed with a **LOCAL PROGRAM** that applies only to the 2 axes of the controller.

The type of program, global program or local program, will affect the command syntax. In a global program, all axes have a unique axis designator (A-H). In a local program, each program addresses only the 2 axes of the controller. These two axes are always referred to as A and B.

The following sections in this chapter discuss each aspect of creating an applications program. Where applicable, subjects are identified as applicable only to LOCAL PROGRAMS with the word LOCAL next to each header.

The program memory size for each DMC-3425 is 80 characters per line and 500 lines long.

Entering Programs

The DMC-3425 has an internal editor that may be used to create and edit programs in the controller's memory. The internal editor is a rudimentary editor and is only recommended when operating with Galil's DOS utilities or through a simple RS-232 communication interface such as the Windows Utility Hyperterminal.

The internal editor is opened by the command ED. Note that the command ED will not open the internal editor if issued from Galil's Window based software - in this case, a Windows based editor will be automatically opened. The Windows based editor provides much more functionality and ease-of-use, therefore, the internal editor is most useful when using a simple terminal with the controller and a Windows based editor is not available.

Once the ED command has been given, each program line is automatically numbered sequentially starting with 000. If no parameter follows the ED command, the editor prompter will default to the last line of the last program in memory. If desired, the user can edit a specific line number or label by specifying a line number or label following ED.

Instruction	Interpretation
:ED	Puts Editor at end of last program
:ED 5	Puts Editor at line 5
:ED #BEGIN	Puts Editor at label #BEGIN

Line numbers appear as 000,001,002 and so on. Program commands are entered following the line numbers. Multiple commands may be given on a single line as long as the total number of characters doesn't exceed 80 characters per line.

While in the Edit Mode, the programmer has access to special instructions for saving, inserting and deleting program lines. These special instructions are listed below:

Edit Mode Commands

<RETURN>

Typing the return key causes the current line of entered instructions to be saved. The editor will automatically advance to the next line. Thus, hitting a series of <RETURN> will cause the editor to advance a series of lines. Note, changes on a program line will not be saved unless a <return> is given.

<cntrl>P

The <cntrl>P command moves the editor to the previous line.

<cntrl>I

The <cntrl>I command inserts a line above the current line. For example, if the editor is at line number 2 and <cntrl>I is applied, a new line will be inserted between lines 1 and 2. This new line will be labeled line 2. The old line number 2 is renumbered as line 3.

<cntrl>D

The <cntrl>D command deletes the line currently being edited. For example, if the editor is at line number 2 and <cntrl>D is applied, line 2 will be deleted. The previous line number 3 is now renumbered as line number 2.

<cntrl>Q

The <cntrl>Q quits the editor mode. In response, the DMC-3425 will return a colon.

After the Edit session is over, the user may list the entered program using the LS command. If no operand follows the LS command, the entire program will be listed. The user can start listing at a specific line or label using the operand n. A command and new line number or label following the start listing operand specifies the location at which listing is to stop.

Example:

Instruction	Interpretation
:LS	List entire program
:LS 5	Begin listing at line 5
:LS 5,9	List lines 5 thru 9
:LS #A,9	List line label #A thru line 9
:LS #A, #A +5	List line label #A and additional 5 lines

Program Format

A DMC program consists of instructions combined to solve a machine control application. Action instructions, such as starting and stopping motion, are combined with Program Flow instructions to form the complete program. Program Flow instructions evaluate real-time conditions, such as elapsed time or motion complete, and alter program flow accordingly.

Each DMC-3425 instruction in a program must be separated by a delimiter. Valid delimiters are the semicolon (;) or carriage return. The semicolon is used to separate multiple instructions on a single program line where the maximum number of instructions on a line is limited by 80 characters. A carriage return enters the final command on a program line.

Using Labels in Programs

All DMC-3425 programs must begin with a label and end with an End (EN) statement. Labels start with the pound (#) sign followed by a maximum of seven characters. The first character must be a letter; after that, numbers are permitted. Spaces are not permitted.

The maximum number of labels that may be defined is 254.

Valid labels

#BEGIN

#SQUARE

#X1

#BEGIN1

Invalid labels

#1Square

#123

Example

Instruction	Interpretation
#START	Beginning of the Program
PR 10000,20000	Specify relative distances on A and B axes
BG AB	Begin Motion

AM	Wait for motion complete
WT 2000	Wait 2 sec
JP #START	Jump to label START
EN	End of Program

The above program moves A and B 10000 and 20000 units. After the motion is complete, the motors rest for 2 seconds. The cycle repeats indefinitely until the stop command is issued.

Special Labels

The DMC-3425 has some special labels, which are used to define input interrupt subroutines, limit switch subroutines, error handling subroutines, command error subroutines and auto start and recovery routines.

#AUTO	Label for automatic execution of program upon power up. This program must be saved in the non-volatile memory with the BP command.
#AUTOERR	Label for detecting errors in the #AUTO routine. If a Checksum error were to occur, the #AUTO would fail to start at power up. This #AUTOERR routine would be called instead.
#ININT	Label for Input Interrupt subroutine
#LIMSWI	Label for Limit Switch subroutine
#POSERR	Label for excess Position Error subroutine
#MCTIME	Label for timeout on Motion Complete trip point
#CMDERR	Label for incorrect command subroutine
#COMINT	Label for communication interrupt on the aux. serial port
#TCPERR	Ethernet communication error

Commenting Programs

There are two methods for commenting programs. The first method uses the NO command and allows for comments to be embedded into Galil programs. The second method used the REM statement and requires the use of Galil software.

NO Command and the Apostrophe (')

Programs on the DMC-3425 can be commented using the command, NO, or the apostrophe. These commands allow the user to include up to 78 characters on a single line. This can be used to include comments from the programmer as in the following example:

Instruction	**Interpretation**
#PATH	Label
NO 2-D CIRCULAR PATH	Comment - No Operation
VMAB	Vector Mode
NO VECTOR MOTION ON A AND B	Comment - No Operation
VS 10000	Vector Speed
NO VECTOR SPEED IS 10000	Comment - No Operation
VP -4000,0	Vector Position
NO BOTTOM LINE	Comment - No Operation
CR 1500,270,-180	Circle Motion
NO HALF CIRCLE MOTION	Comment - No Operation

VP 0,3000	Vector Position
‘ TOP LINE	Comment - No Operation
CR 1500,90,-180	Circle
‘ HALF CIRCLE MOTION	Comment - No Operation
VE	Vector End
‘ END VECTOR SEQUENCE	Comment - No Operation
BGS	Begin Sequence
‘ BEGIN SEQUENCE MOTION	Comment - No Operation
EN	End of Program
‘ END OF PROGRAM	Comment - No Operation

NOTE: NO and the apostrophe are controller commands. Therefore, inclusion of these commands will require a small process time by the controller.

REM Command

If you are using Galil software to communicate with the DMC-3425 controller, you may also include REM statements. ‘REM’ statements begin with the word ‘REM’ and may be followed by any comments that are on the same line. The Galil terminal software will remove these statements when the program is downloaded to the controller. For example:

```
#PATH
REM 2-D CIRCULAR PATH
VMAB
REM VECTOR MOTION ON A AND B
VS 10000
REM VECTOR SPEED IS 10000
VP -4000,0
REM BOTTOM LINE
CR 1500,270,-180
REM HALF CIRCLE MOTION
VP 0,3000
REM TOP LINE
CR 1500,90,-180
REM HALF CIRCLE MOTION
VE
REM END VECTOR SEQUENCE
BGS
REM BEGIN SEQUENCE MOTION
EN
REM END OF PROGRAM
```

These REM statements will be removed when this program is downloaded to the controller.

Executing Programs - Multitasking

The DMC-3425 can run 2 independent programs simultaneously. These programs are called threads and are numbered 0 and 1, where 0 is the main thread. Multitasking is useful for executing independent operations such as PLC functions that occur independently of motion.

The main thread differs from the others in the following ways:

1. Only the main thread, thread 0, may use the input command, IN.

2. When automatic subroutines are implemented for limit switches, position errors or command errors, they are executed in thread 0.

To begin execution of the various programs, use the following instruction:

XQ #A, n

Where n indicates the thread number. To halt the execution of any thread, use the instruction

HX n

where n is the thread number.

Note that both the XQ and HX commands can be performed by an executing program.

The example below produces a waveform on Output 1 independent of a move.

Instruction	**Interpretation**
#TASK1	Task1 label
AT0	Initialize reference time
CB1	Clear Output 1
#LOOP1	Loop1 label
AT 10	Wait 10 msec from reference time
SB1	Set Output 1
AT –40	Wait 40 msec from reference, then initialize reference
CB1	Clear Output 1
JP #LOOP1	Repeat Loop1
#TASK0	Task2 label
XQ #TASK1,1	Execute Task1
#LOOP2	Loop2 label
PR 1000	Define relative distance
BGA	Begin motion
AMA	After motion done
WT 10	Wait 10 msec
JP #LOOP2,@IN[2]=1	Repeat motion unless Input 2 is low
HX	Halt all tasks
EN	End of Program

The program above is executed with the instruction XQ #TASK0,0 which designates TASK0 as the main thread (i.e. Thread 0). #TASK1 is executed within TASK0.

Debugging Programs

The DMC-3425 provides commands and operands that are useful in debugging application programs. These commands include interrogation commands to monitor program execution, determine the state of the controller and the contents of the controllers program, array, and variable space. Operands also contain important status information that can help to debug a program. Breakpoint and single stepping commands are available to actively debug a program while in operation.

Trace Command

The trace command causes the controller to send each line in a program to the host computer immediately prior to execution. Tracing is enabled with the command, TR1. TR0 turns the trace function off.

NOTE: When the trace function is enabled, the line numbers as well as the command line will be displayed as each command line is executed.

Error Code Command

When there is a program error, the DMC-3425 halts the program execution at the point where the error occurs. To display the last line number of program execution, issue the command, MG _ED.

The user can obtain information about the type of error condition that occurred by using the command, TC1. This command reports back a number and a text message that describes the error condition. The command, TC0 or TC, will return the error code without the text message. For more information about the command, TC, see the Command Reference.

Example

The following program has an error. It attempts to specify a relative movement while the A-axis is already in motion. When the program is executed, the controller stops at line 003. The user can then query the controller using the command, TC1. The controller responds with the corresponding explanation:

Instruction	Interpretation
:ED	Edit Mode
000 #A	Program Label
001 PR1000	Position Relative 1000
002 BGA	Begin
003 PR5000	Position Relative 5000
004 EN	End
<cntrl> Q	Quit Edit Mode
:XQ #A	Execute #A
?003 PR5000	Error on Line 3
:TC1	Tell Error Code
?7 Command not valid while running.	Command not valid while running
:ED 3	Edit Line 3
003 AMA;PR5000;BGA	Add After Motion Done
<cntrl> Q	Quit Edit Mode
:XQ #A	Execute #A

Stop Code Command

The status of motion for each axis can be determined by using the stop code command, SC. This can be useful when motion on an axis has stopped unexpectedly. The command SC will return a number representing the motion status. See the command reference for further information.

RAM Memory Interrogation Commands

For debugging the status of the program memory, array memory, or variable memory, the DMC-3425 has several useful commands. The command, DM ?, will return the number of array elements

currently available. The command, DA?, will return the number of arrays which can be currently defined. The DMC-3425 will have a maximum of 2000 array elements in up to 14 arrays. If an array of 100 elements is defined, the command DM? will return the value 1900 and the command DA? will return 13.

To list the contents of the variable space, use the interrogation command LV (List Variables). To list the contents of array space, use the interrogation command, LA (List Arrays). To list the contents of the Program space, use the interrogation command, LS (List). To list the application program labels only, use the interrogation command, LL (List Labels).

Operands

An operand is a value in the controller. Below is a list of specific operands that are particularly valuable for program debugging. To display an operand, the message command may be used. For example, since the operand, _ED contains the last line of program execution, the command MG _ED will display this line number.

_ED contains the last line of program execution. Useful to determine where program stopped.

_DL contains the number of available labels.

_UL contains the number of available variables.

_DA contains the number of available arrays.

_DM contains the number of available array elements.

_AB contains the state of the Abort Input

_LFx contains the state of the forward limit switch for the 'x' axis

_LRx contains the state of the reverse limit switch for the 'x' axis

Breakpoints and single stepping

The DMC-3425 has commands which allow active debugging of programs. The BK command is a breakpoint which may be set to trigger upon execution of a specified line and thread. Upon the program executing the specified line, the program or thread will pause at that line. The SL command may then be used to single step through the program from that breakpoint. See the command reference for more information on the BK and SL command.

EEPROM Memory Interrogation Operands

When the DMC-3425 powers up, any data stored in the EEPROM memory is automatically loaded for use. This data includes the user program, variables and arrays, and controller parameters. If the EEPROM has been corrupted, the corresponding memory sector is flagged as in error. The operand _RS contains the state of the EEPROM as follows:

Bit	Error Condition
Bit 3	Master reset error
Bit 2	Program checksum error
Bit 1	Parameter checksum error
Bit 0	Variable checksum error

Program Flow Commands

The DMC-3425 provides instructions to control program flow. The DMC-3425 program sequencer normally executes program instructions sequentially. The program flow can be altered with the use of event triggers, trippoints, and conditional jump statements.

Event Triggers & Trippoints

To function independently from the host computer, the DMC-3425 can be programmed to make decisions based on the occurrence of an event. Such events include waiting for motion to be complete, waiting for a specified amount of time to elapse, or waiting for an input to change logic levels.

The DMC-3425 provides several event triggers that cause the program sequencer to halt until the specified event occurs. Normally, a program is automatically executed sequentially one line at a time. When an event trigger instruction is decoded, however, the actual program sequence is halted. The program sequence does not continue until the event trigger is "tripped". For example, the motion complete trigger can be used to separate two move sequences in a program. The commands for the second move sequence will not be executed until the motion is complete on the first motion sequence. In this way, the DMC-3425 can make decisions based on its own status or external events without intervention from a host computer.

NOTE: Event triggers should only be used within a program and not sent to the controller as a direct command.

DMC-3425 Event Triggers

Command	Function
AM A B C D E F G H or S	Halts program execution until motion is complete on the specified axes or motion sequence(s). AM with no parameter tests for motion complete on all axes. This command is useful for separating motion sequences in a program.
AD A or B or C or D or E or F or G or H	Halts program execution until position command has reached the specified relative distance from the start of the move. Only one axis may be specified at a time.
AR A or B or C or D or E or F or G or H	Halts program execution until after specified distance from the last AR or AD command has elapsed. Only one axis may be specified at a time.
AP A or B or C or D or E or F or G or H	Halts program execution until after absolute position occurs. Only one axis may be specified at a time.
MF A or B or C or D or E or F or G or H	Halt program execution until after forward motion reached absolute position. Only one axis may be specified. If position is already past the point, then MF will trip immediately. Will function on geared axis or aux. inputs.
MR A or B or C or D or E or F or G or H	Halt program execution until after reverse motion reached absolute position. Only one axis may be specified. If position is already past the point, then MR will trip immediately. Will function on geared axis or aux. inputs.
MC A or B or C or D or E or F or G or H	Halt program execution until after the motion profile has been completed and the encoder has entered or passed the specified position. TW a,b,c,d sets timeout to declare an error if not in position. If timeout occurs, then the trippoint will clear and the stopcode will be set to 99. An application program will jump to label #MCTIME.
AI +/- n	Halts program execution until after specified input is at specified logic level. n specifies input line. Positive is high logic level; negative is low level.
AS A B C D E F G H	Halts program execution until specified axis has reached its slew speed.
AT +/-n	Halts program execution until n msec from reference time. AT 0 sets reference. AT n waits n msec from reference. AT -n waits n msec from reference and sets new reference after elapsed time.
AV n	Halts program execution until specified distance along a coordinated path has occurred.
WT n	Halts program execution until specified time in msec has elapsed.

Example- Multiple Move Sequence

The AM trippoint is used to separate the two PR moves. If AM is not used, the controller returns a ? for the second PR command because a new PR cannot be given until motion is complete.

Instruction	Interpretation
#TWOMOVE	Label
PR 2000	Position Command
BGA	Begin Motion
AMA	Wait for Motion Complete
PR 4000	Next Position Move
BGA	Begin 2nd move
EN	End program

Example- Set Output after Distance

Set output bit 1 after a distance of 1000 counts from the start of the move. The accuracy of the trippoint is the speed multiplied by the sample period.

Instruction	Interpretation
#SETBIT	Label
SP 10000	Speed is 10000
PA 20000	Specify Absolute position
BGA	Begin motion
AD 1000	Wait until 1000 counts
SB1	Set output bit 1
EN	End program

Example- Repetitive Position Trigger

To set the output bit every 10000 counts during a move, the AR trippoint is used as shown in the next example.

Instruction	Interpretation
#TRIP	Label
JG 50000	Specify Jog Speed
BGA;n=0	Begin Motion
#REPEAT	# Repeat Loop
AR 10000	Wait 10000 counts
TPA	Tell Position
SB1	Set output 1
WT50	Wait 50 msec
CB1	Clear output 1
n=n+1	Increment counter
JP #REPEAT,n<5	Repeat 5 times
STA	Stop
EN	End

Example - Start Motion on Input

This example waits for input 1 to go low and then starts motion.

NOTE: The AI command actually halts execution of the program until the input occurs. If you do not want to halt the program sequences, you can use the Input Interrupt function (II) or use a conditional jump on an input, such as JP #GO,@IN[1] = 1.

Instruction	Interpretation
#INPUT	Program Label
AI-1	Wait for input 1 low
PR 10000	Position command
BGA	Begin motion
EN	End program

Example - Set Output when At Speed

Instruction	Interpretation
#ATSPEED	Program Label
JG 50000	Specify jog speed
AC 10000	Acceleration rate
BGA	Begin motion
ASA	Wait for at slew speed 50000
SB1	Set output 1
EN	End program

Example - Change Speed along Vector Path

The following program changes the feedrate or vector speed at the specified distance along the vector. The vector distance is measured from the start of the move or from the last AV command.

Instruction	Interpretation
#VECTOR	Label
VMAB;VS 5000	Coordinated path
VP 10000,20000	Vector position
VP 20000,30000	Vector position
VE	End vector
BGS	Begin sequence
AV 5000	After vector distance
VS 1000	Reduce speed
EN	End

Example - Multiple Move with Wait

This example makes multiple relative distance moves by waiting for each to be complete before executing new moves.

Instruction	Interpretation
#MOVES	Label
PR 12000	Distance
SP 20000	Speed
AC 100000	Acceleration
BGA	Start Motion
AD 10000	Wait a distance of 10,000 counts
SP 5000	New Speed
AMA	Wait until motion is completed
WT 200	Wait 200 ms

PR -10000	New Position
SP 30000	New Speed
AC 150000	New Acceleration
BGA	Start Motion
EN	End

Example- Define Output Waveform Using AT

The following program causes Output 1 to be high for 10 msec and low for 40 msec. The cycle repeats every 50 msec.

Instruction	**Interpretation**
#OUTPUT	Program label
AT0	Initialize time reference
SB1	Set Output 1
#LOOP	Loop
AT 10	After 10 msec from reference,
CB1	Clear Output 1
AT -40	Wait 40 msec from reference and reset reference
SB1	Set Output 1
JP #LOOP	Loop
EN	

Conditional Jumps

The DMC-3425 provides Conditional Jump (JP) and Conditional Jump to Subroutine (JS) instructions for branching to a new program location based on a specified condition. The conditional jump determines if a condition is satisfied and then branches to a new location or subroutine. Unlike event triggers, the conditional jump instruction does not halt the program sequence. Conditional jumps are useful for testing events in real-time. They allow the DMC-3425 to make decisions without a host computer. For example, the DMC-3425 can decide between two motion profiles based on the state of an input line.

Command Format - JP and JS

Format:	**Description**
JS destination, logical condition	Jump to subroutine if logical condition is satisfied
JP destination, logical condition	Jump to location if logical condition is satisfied

The destination is a program line number or label where the program sequencer will jump if the specified condition is satisfied. Note that the line number of the first line of program memory is 0. The comma designates "IF". The logical condition tests two operands with logical operators.

Logical operators:

OPERATOR	DESCRIPTION
<	less than
>	greater than
=	equal to
<=	less than or equal to
>=	greater than or equal to
<>	not equal

Conditional Statements

The conditional statement is satisfied if it evaluates to any value other than zero. The conditional statement can be any valid DMC-3425 numeric operand, including variables, array elements, numeric values, functions, keywords, and arithmetic expressions. If no conditional statement is given, the jump will always occur.

Number	V1=6
Numeric Expression	V1=V7*6
	@ABS[V1]>10
Array Element	V1<Count[2]
Variable	V1<V2
Internal Variable	_TPA=0
	_TVA>500
I/O	V1>@AN[2]
	@IN[1]=0

Multiple Conditional Statements

The DMC-3425 will accept multiple conditions in a single jump statement. The conditional statements are combined in pairs using the operands "&" and "|". The "&" operand between any two conditions, requires that both statements must be true for the combined statement to be true. The "|" operand between any two conditions, requires that only one statement be true for the combined statement to be true.

NOTE: Each condition must be placed in parentheses for proper evaluation by the controller. In addition, the DMC-3425 executes operations from left to right. For further information on Mathematical Expressions and the bit-wise operators '&' and '|', see pg. 127.

Example using variables named V1, V2, V3 and V4:

```
JP #TEST, (V1<V2) & (V3<V4)
```

In this example, this statement will cause the program to jump to the label #TEST if V1 is less than V2 and V3 is less than V4. To illustrate this further, consider this same example with an additional condition:

```
JP #TEST, ((V1<V2) & (V3<V4)) | (V5<V6)
```

This statement will cause the program to jump to the label #TEST under two conditions; 1. If V1 is less than V2 and V3 is less than V4. OR 2. If V5 is less than V6.

Examples

If the condition for the JP command is satisfied, the controller branches to the specified label or line number and continues executing commands from this point. If the condition is not satisfied, the controller continues to execute the next commands in sequence.

Instruction	Interpretation
JP #Loop, COUNT<10	Jump to #Loop if the variable, COUNT, is less than 10
JS #MOVE2,@IN[1]=1	Jump to subroutine #MOVE2 if input 1 is logic level high. After the subroutine MOVE2 is executed, the program sequencer returns to the main program location where the subroutine was called.
JP #BLUE,@ABS[V2]>2	Jump to #BLUE if the absolute value of variable, V2, is greater than 2
JP #C,V1*V7<=V8*V2	Jump to #C if the value of V1 times V7 is less than or equal to the value of V8*V2
JP#A	Jump to #A

Move the A motor to absolute position 1000 counts and back to zero ten times. Wait 100 msec between moves.

Instruction	Interpretation
#BEGIN	Begin Program
COUNT=10	Initialize loop counter
#LOOP	Begin loop
PA 1000	Position absolute 1000
BGA	Begin move
AMA	Wait for motion complete
WT 100	Wait 100 msec
PA 0	Position absolute 0
BGA	Begin move
AMA	Wait for motion complete
WT 100	Wait 100 msec
COUNT=COUNT-1	Decrement loop counter
JP #LOOP,COUNT>0	Test for 10 times thru loop
EN	End Program

If, Else, and Endif

The DMC-3425 provides a structured approach to conditional statements using IF, ELSE and ENDIF commands.

Using the IF and ENDIF Commands

An IF conditional statement is formed by the combination of an IF and ENDIF command. The IF command has as its arguments one or more conditional statements. If the conditional statement(s) evaluates true, the command interpreter will continue executing commands which follow the IF command. If the conditional statement evaluates false, the controller will ignore commands until the associated ENDIF command is executed OR an ELSE command occurs in the program (see discussion of ELSE command below).

NOTE: An ENDIF command must always be executed for every IF command that has been executed. It is recommended that the user not include jump commands inside IF conditional statements since this causes re-direction of command execution. In this case, the command interpreter may not execute an ENDIF command.

Using the ELSE Command

The ELSE command is an optional part of an IF conditional statement and allows for the execution of command only when the argument of the IF command evaluates False. The ELSE command must occur after an IF command and has no arguments. If the argument of the IF command evaluates false, the controller will skip commands until the ELSE command. If the argument for the IF command evaluates true, the controller will execute the commands between the IF and ELSE command.

Nesting IF Conditional Statements

The DMC-3425 allows for IF conditional statements to be included within other IF conditional statements. This technique is known as 'nesting' and the DMC-3425 allows up to 255 IF conditional statements to be nested. This is a very powerful technique allowing the user to specify a variety of different cases for branching.

Command Format - IF, ELSE and ENDIF

Format:	**Description**
IF <condition>	Execute commands proceeding IF command (up to ELSE command) if conditional statement(s) is true, otherwise continue executing at ENDIF command or optional ELSE command.
ELSE	Optional command. Allows for commands to be executed when argument of IF command evaluates not true. Can only be used with IF command.
ENDIF	Command to end IF conditional statement. Program must have an ENDIF command for every IF command.

Instruction	**Interpretation**
#TEST	Begin Main Program "TEST"
II,,3	Enable interrupts on input 1 and input 2
MG "WAITING FOR INPUT 1, INPUT 2"	Output message
#LOOP	Label to be used for endless loop
JP #LOOP	Endless loop
EN	End of main program
#ININT	Input Interrupt Subroutine
IF (@IN[1]=0)	IF conditional statement based on input 1
IF (@IN[2]=0)	2nd IF executed if 1st IF conditional true
MG "INPUT 1 AND INPUT 2 ARE ACTIVE"	Message executed if 2nd IF is true
ELSE	ELSE command for 2nd IF statement
MG "ONLY INPUT 1 IS ACTIVE	Message executed if 2nd IF is false
ENDIF	End of 2nd conditional statement
ELSE	ELSE command for 1st IF statement
MG"ONLY INPUT 2 IS ACTIVE"	Message executed if 1st IF statement
ENDIF	End of 1st conditional statement
#WAIT	Label to be used for a loop

JP#WAIT,(@IN[1]=0) | (@IN[2]=0)	Loop until Input 1& 2 are not active
RI0	End Input Interrupt Routine without restoring trippoints

Subroutines

A subroutine is a group of instructions beginning with a label and ending with an end command (EN). Subroutines are called from the main program with the jump subroutine instruction JS, followed by a label or line number, and conditional statement. Up to 8 subroutines can be nested. After the subroutine is executed, the program sequencer returns to the program location where the subroutine was called unless the subroutine stack is manipulated as described in the following section.

An example of a subroutine to draw a square 500 counts per side is given below. The square is drawn at vector position 1000,1000.

Instruction	**Interpretation**
#M	Begin Main Program
CB1	Clear Output Bit 1 (pick up pen)
VP 1000,1000;LE;BGS	Define vector position; move pen
AMS	Wait for after motion trippoint
SB1	Set Output Bit 1 (put down pen)
JS #Square;CB1	Jump to square subroutine
EN	End Main Program
#Square	Square subroutine
V1=500;JS #L	Define length of side
V1=-V1;JS #L	Switch direction
EN	End subroutine
#L;PR V1,V1;BGA	Define A,B; Begin A
AMA;BGB;AMA	After motion on A, Begin B
EN	End subroutine

Stack Manipulation

It is possible to manipulate the subroutine stack by using the ZS command. Every time a JS instruction, interrupt or automatic routine (such as #POSERR or #LIMSWI) is executed, the subroutine stack is incremented by 1. Normally the stack is restored with an EN instruction. Occasionally it is desirable not to return back to the program line where the subroutine or interrupt was called. The ZS1 command clears 1 level of the stack. This allows the program sequencer to continue to the next line. The ZS0 command resets the stack to its initial value.

Auto-Start and Auto Error Routine

The DMC-3425 has two special labels for automatic program execution. A program which has been saved into the controllers non-volatile memory can be automatically executed upon power up or reset by beginning the program with the label #AUTO. The program must be saved into non-volatile memory using the command, BP.

If the program loaded onto the EEPROM has a checksum error at power-up, the routine #AUTOERR will run instead, allowing the user to determine the nature of the checksum error. The _RS operand may be used to determine what sector of the EEPROM has been corrupted.

Automatic Subroutines for Monitoring Conditions

Often it is desirable to monitor certain conditions continuously without tying up the host or DMC-3425 program sequences. The DMC-3425 can monitor several important conditions in the background. These conditions include checking for the occurrence of a limit switch, a defined input, position error, or a command error. Automatic monitoring is enabled by inserting a special, predefined label in the applications program, and having an application program actively executing on the controller. The pre-defined labels are:

SUBROUTINE	DESCRIPTION
#LIMSWI	Limit switch on any axis goes low
#ININT	Input specified by II goes low
#POSERR	Position error exceeds limit specified by ER
#MCTIME	Motion Complete timeout occurred. Timeout period set by TW command
#CMDERR	Bad command given
#TCPERR	Ethernet Communication Error

The following examples illustrate the use of the automatic subroutines:

Example - Limit Switch:

This simple program prints a message upon the occurrence of a limit switch. For the #LIMSWI subroutine to execute, the DMC-3425 ***must be executing an applications program from memory*** and ***the controller must be commanding the motor to move***. The RE command is used to return from the #LIMSWI subroutine. The #LIMSWI subroutine will be re-executed if the limit switch remains active.

Instruction	Interpretation
#LOOP	Dummy Program
JP #LOOP;EN	Jump to Loop
#LIMSWI	Limit Switch Label
MG "LIMIT OCCURRED"	Print Message
RE	Return to main program

Example - Position Error

Instruction	Interpretation
:ED	Edit Mode
000 #LOOP	Dummy Program
001 JP #LOOP;EN	Loop
002 #POSERR	Position Error Routine
003 V1=_TEA	Read Position Error
004 MG "EXCESS POSITION ERROR"	Print Message
005 MG "ERROR=",V1=	Print Error
006 RE	Return from Error
<control> Q	Quit Edit Mode
:XQ #LOOP	Execute Dummy Program
:JG 100000	Jog at High Speed
:BGA	Begin Motion

Example - Input Interrupt

This simple program jogs the A and C motors (C motor is the first motor of the first slave controller of a distributed control system). When the first input of the master (input 1), goes low, the controller will stop motion on both axes. When the input returns high, the motors will resume jogging.

Instruction	Interpretation
#A	Label
II1	Input Interrupt on 1
JG 30000,,60000	Jog
BGAD	Begin Motion
#LOOP;JP#LOOP;EN	Loop
#ININT	Input Interrupt
STAD;AMAD	Stop Motion
#TEST;JP #TEST, @IN[1]=0	Test for Input 1 still low
JG 30000,,,6000	Restore Velocities
BGAD	Begin motion
RI0	Return from interrupt routine to Main Program and do not re-enable trippoints

Example - Motion Complete Timeout

This simple program will issue the message "A fell short" if the A axis does not reach the commanded position within 1 second of the end of the profiled move.

Instruction	Interpretation
#BEGIN	Begin main program
TW 1000	Set the time out to 1000 ms
PA 10000	Position Absolute command
BGA	Begin motion
MCA	Motion Complete trip point
EN	End main program
#MCTIME	Motion Complete Subroutine
MG "A fell short"	Send out a message
EN	End subroutine

Example - Command Error

The above program prompts the operator to enter a jog speed. If the operator enters a number out of range (greater than 8 million), the #CMDERR routine will be executed prompting the operator to enter a new number.

In multitasking applications, there is an alternate method for handling command errors from different threads. Using the XQ command along with the special operands described below allows the controller to either skip or retry invalid commands.

Instruction	Interpretation
#BEGIN	Begin main program
IN "ENTER SPEED", SPEED	Prompt for speed
JG SPEED;BGA;	Begin motion
JP #BEGIN	Repeat

EN	End main program
#CMDERR	Command error utility
JP#DONE,_ED<>2	Check if error on line 2
JP#DONE,_TC<>6	Check if out of range
MG "SPEED TOO HIGH"	Send message
MG "TRY AGAIN"	Send message
ZS1	Adjust stack
JP #BEGIN	Return to main program
#DONE	End program if other error
ZS0	Zero stack
EN	End program

OPERAND	FUNCTION
_ED1	Returns the number of the thread that generated an error
_ED2	Retry failed command (operand contains the location of the failed command)
_ED3	Skip failed command (operand contains the location of the command after the failed command)

The operands are used with the XQ command in the following format:

XQ _ED2 (or _ED3),_ED1,1

Where the ",1" at the end of the command line indicates a restart; therefore, the existing program stack will not e removed when the above format executes.

The following example shows an error correction routine that uses the operands.

Example - Command Error w/Multitasking

The following program illustrates a common program problem. In this case, a variable is used as a command argument and the variable is inadvertently set to an illegal value. This simple command error subroutine recognizes the type of error, modifies the variable and continues the program at the point of the error. If the program has an invalid command error, skip the command and continue to execute the program. To demonstrate the program, while the simple loop #A is executing on thread 0 (XQ#A,0), begin execution of the second task, XQ#B,1

Instruction	Interpretation
#A	Begin thread 0 (continuous loop)
JP#A	
EN	End of thread 0
#B	Begin thread 1
KP -1	Set KP to value of N, an invalid value
TY	Issue invalid command
EN	End of thread 1
#CMDERR	Begin command error subroutine
IF(_TC=6)	If error is "Number Out of Range" (-1).
N=1	Set N to a valid number
XQ _ED2,_ED1,1	Retry KP N command
ENDIF	
IF(_TC=1)	If error is "Invalid Command" (TY)

XQ _ED3,_ED1,1	Skip invalid command
ENDIF	
EN	End of command error routine

Example – Ethernet Communication Error

This simple program executes in the DMC-3425 and indicates (via the serial port) when a communication handle fails. By monitoring the serial port, the user can re-establish communication if needed.

Instruction	**Interpretation**
#LOOP	Simple program loop
JP#LOOP	
EN	
#TCPERR	Ethernet communication error auto routine
MG {P1}_IA4	Send message to serial port indicating which handle did not receive proper acknowledgment
RE	

Mathematical and Functional Expressions

Mathematical Operators

For manipulation of data, the DMC-3425 provides the use of the following mathematical operators:

Operator	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division
&	Logical And (Bit-wise)
|	Logical Or (On some computers, a solid vertical line appears as a broken line)
()	Parenthesis

The numeric range for addition, subtraction and multiplication operations is +/-2,147,483,647.9999. The precision for division is 1/65,000.

Mathematical operations are executed from left to right. Calculations within parentheses have precedence.

SPEED=7.5*V1/2	The variable, SPEED, is equal to 7.5 multiplied by V1 and divided by 2
COUNT=COUNT+2	The variable, COUNT, is equal to the current value plus 2.
RESULT=_TPA-(@COS[45]*40)	Puts the position of A - 28.28 in RESULT. 40 * cosine of 45° is 28.28
TEMP=@IN[1]&@IN[2]	TEMP is equal to 1 only if Input 1 and Input 2 are high

Bit-Wise Operators

The mathematical operators & and | are bit-wise operators. The operator, &, is a Logical And. The operator, |, is a Logical Or. These operators allow for bit-wise operations on any valid DMC-3425 numeric operand, including variables, array elements, numeric values, functions, keywords, and arithmetic expressions. The bit-wise operators may also be used with strings. This is useful for separating characters from an input string. When using the input command for string input, the input variable will hold up to 6 characters. These characters are combined into a single value that is represented as 32 bits of integer and 16 bits of fraction. Each ASCII character is represented as one byte (8 bits), therefore the input variable can hold up to six characters. The first character of the string will be placed in the top byte of the variable and the last character will be placed in the lowest significant byte of the fraction. The characters can be individually separated by using bit-wise operations as illustrated in the following example:

Instruction	**Interpretation**
#TEST	Begin main program
IN "ENTER",LEN{S6}	Input character string of up to 6 characters into variable 'LEN'
FLEN=@FRAC[LEN]	Define variable 'FLEN' as fractional part of variable 'LEN'
FLEN=$10000*FLEN	Shift FLEN by 32 bits (IE - convert fraction, FLEN, to integer)
LEN1=(FLEN&$00FF)	Mask top byte of FLEN and set this value to variable 'LEN1'
LEN2=(FLEN&$FF00)/$100	Let variable, 'LEN2' = top byte of FLEN
LEN3=LEN&$000000FF	Let variable, 'LEN3' = bottom byte of LEN
LEN4=(LEN&$0000FF00)/$100	Let variable, 'LEN4' = second byte of LEN
LEN5=(LEN&$00FF0000)/$10000	Let variable, 'LEN5' = third byte of LEN
LEN6=(LEN&$FF000000)/$1000000	Let variable, 'LEN6' = fourth byte of LEN
MG LEN6 {S4}	Display 'LEN6' as string message of up to 4 chars
MG LEN5 {S4}	Display 'LEN5' as string message of up to 4 chars
MG LEN4 {S4}	Display 'LEN4' as string message of up to 4 chars
MG LEN3 {S4}	Display 'LEN3' as string message of up to 4 chars
MG LEN2 {S4}	Display 'LEN2' as string message of up to 4 chars
MG LEN1 {S4}	Display 'LEN1' as string message of up to 4 chars
EN	

This program will accept a string input of up to 6 characters, parse each character, and then display each character. Notice also that the values used for masking are represented in hexadecimal (as denoted by the preceding '$'). For more information, see section *Sending Messages.*

To illustrate further, if the user types in the string "TESTME" at the input prompt, the controller will respond with the following:

T	Response from command MG LEN6 {S4}
E	Response from command MG LEN5 {S4}
S	Response from command MG LEN4 {S4}
T	Response from command MG LEN3 {S4}
M	Response from command MG LEN2 {S4}
E	Response from command MG LEN1 {S4}

Functions

FUNCTION	DESCRIPTION
@SIN[n]	Sine of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution)
@COS[n]	Cosine of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution)
@TAN[n]	Tangent of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution)
@ASIN*[n]	Arc Sine of n, between -90° and +90°. Angle resolution in 1/64000 degrees.
@ACOS* [n}	Arc Cosine of n, between 0 and 180°. Angle resolution in 1/64000 degrees.
@ATAN* [n]	Arc Tangent of n, between -90° and +90°. Angle resolution in 1/64000 degrees
@COM[n]	1's Complement of n
@ABS[n]	Absolute value of n
@FRAC[n]	Fraction portion of n
@INT[n]	Integer portion of n
@RND[n]	Round of n (Rounds up if the fractional part of n is .5 or greater)
@SQR[n]	Square root of n (Accuracy is +/-.0001)
@IN[n]	Return digital input at general input n (where n starts at 1)
@OUT[n]	Return digital output at general output n (where n starts at 1)
@AN[n]	Return analog input at general analog in n (where n starts at 1)

* Note that these functions are multi-valued. An application program may be used to find the correct band.

Functions may be combined with mathematical expressions. The order of execution of mathematical expressions is from left to right and can be over-ridden by using parentheses.

Instruction	Interpretation
V1=@ABS[V7]	The variable, V1, is equal to the absolute value of variable V7.
V2=5*@SIN[POS]	The variable, V2, is equal to five times the sine of the variable, POS.
V3=@IN[1]	The variable, V3, is equal to the digital value of input 1.
V4=2*(5+@AN[5])	The variable, V4, is equal to the value of analog input 5 plus 5, then multiplied by 2.

Variables

For applications that require a parameter that is variable, the DMC-3425 provides 126 variables. These variables can be numbers or strings. A program can be written in which certain parameters, such as position or speed, are defined as variables. The variables can later be assigned by the operator or determined by program calculations. For example, a cut-to-length application may require that a cut length be variable.

Instruction	Interpretation
PR POSA	Assigns variable POSA to PR command
JG RPMB70	Assigns variable RPMB multiplied by 70 to JG command.

Programmable Variables

The DMC-3425 allows the user to create up to 126 variables. Each variable is defined by a name that can be up to eight characters. The name must start with an alphabetic character, however, numbers are permitted in the rest of the name. Spaces are not permitted. Variable names should not be the same as DMC-3425 instructions. For example, PR is not a good choice for a variable name.

Examples of valid and invalid variable names are:

Valid Variable Names

POSA

POS1

SPEEDC

Invalid Variable Names

REALLONGNAME	; Cannot have more than 8 characters
123	; Cannot begin variable name with a number
SPEED C	; Cannot have spaces in the name

Assigning Values to Variables

Assigned values can be numbers, internal variables and keywords, functions, controller parameters and strings;

The range for numeric variable values is 4 bytes of integer (2^{31}) followed by two bytes of fraction

(+/-2,147,483,647.9999).

Numeric values can be assigned to programmable variables using the equal sign.

Any valid DMC-3425 function can be used to assign a value to a variable. For example, V1=@ABS[V2] or V2=@IN[1]. Arithmetic operations are also permitted.

To assign a string value, the string must be in quotations. String variables can contain up to six characters that must be in quotation.

Instruction	**Interpretation**
POSX=_TPA	Assigns returned value from TPA command to variable POSX.
SPEED=5.75	Assigns value 5.75 to variable SPEED
INPUT=@IN[2]	Assigns logical value of input 2 to variable INPUT
V2=V1+V3*V4	Assigns the value of V1 plus V3 times V4 to the variable V2.
VAR="CAT"	Assign the string, CAT, to VAR

Assigning Variable Values to Controller Parameters

Variable values may be assigned to controller parameters such as GN or PR.

PR V1	Assign V1 to PR command
SP VS*2000	Assign VS*2000 to SP command

Displaying the value of variables at the terminal

Variables may be sent to the screen using the format, variable=. For example, V1= , returns the value of the variable V1.

Example - Using Variables for Joystick

The example below reads the voltage of an A-B joystick and assigns it to variables VA and VB to drive the motors at proportional velocities, where

10 Volts = 3000 rpm = 200000 c/sec

Speed/Analog input = 200000/10 = 20000

Instruction	Interpretation
#JOYSTIK	Label
JG 0,0	Set in Jog mode
BGAB	Begin Motion
#LOOP	Loop
VX=@AN[1]*20000	Read joystick A
VY=@AN[2]*20000	Read joystick B
JG VA,VB	Jog at variable VA,VB
JP#LOOP	Repeat
EN	End

Operands

Operands allow motion or status parameters of the DMC-3425 to be incorporated into programmable variables and expressions. Most DMC-3425 commands have an equivalent operand - which are designated by adding an underscore (_) prior to the DMC-3425 command. The command reference indicates which commands have an associated operand.

Status commands such as Tell Position return actual values, whereas action commands such as KP or SP return the values in the DMC-3425 registers. The axis designation is required following the command.

Instruction	Interpretation
POSA=_TPA	Assigns value from Tell Position A to the variable POSA.
VAR1=_KPA*2	Assigns value from KPA multiplied by two to variable, VAR1.
JP #LOOP,_TEA>5	Jump to #LOOP if the position error of A is greater than 5
JP #ERROR,_TC=1	Jump to #ERROR if the error code equals 1.

Operands can be used in an expression and assigned to a programmable variable, but they cannot be assigned a value. For example: _TPA=2 is invalid.

Special Operands

The DMC-3425 provides a few additional operands that give access to internal variables that are not accessible by standard DMC-3425 commands.

Operand	Function
_BGn	*Returns a 1 if motion on axis ‘n’ is complete, otherwise returns 0.
_BN	*Returns serial # of the board.
_DA	*Returns the number of arrays available
_DL	*Returns the number of available labels for programming
_DM	*Returns the available array memory

_HMn	*Returns status of Home Switch (equals 0 or 1)
_LFn	Returns status of Forward Limit switch input of axis 'n' (equals 0 or 1)
_LRn	Returns status of Reverse Limit switch input of axis 'n' (equals 0 or 1)
_UL	*Returns the number of available variables
TIME	Free-Running Real Time Clock (off by 2.4% - Resets with power-on). **NOTE:** TIME does not use an underscore character (_) as other operands.

* These operands have corresponding commands while the operands _LF, _LR and TIME do not have any associated commands. All operands are listed in the Command Reference Manual.

Examples

V1=_LFA	Assign V1 the state of the Forward Limit Switch on the A-axis
V3=TIME	Assign V3 the current value of the time clock
V4=_HMD	Assign V4 the logical state of the Home input on the D-axis

Arrays

For storing and collecting numerical data, the DMC-3425 provides array space for 2000 elements. The arrays are one-dimensional and up to 14 different arrays may be defined. The array data is available to both threads on each controller. When operating with multiple controllers, arrays are only defined within the same controller.

Each array element has a numeric range of 4 bytes of integer (2^{31}) followed by two bytes of fraction (+/-2,147,483,647.9999).

Arrays can be used to capture real-time data, such as position, torque and analog input values. In the contouring mode, arrays are convenient for holding the points of a position trajectory in a record and playback application.

Defining Arrays

An array is defined with the command DM. The user must specify a name and the number of entries to be held in the array. An array name can contain up to eight characters, starting with an uppercase alphabetic character. The number of entries in the defined array is enclosed in [ ].

DM POSA[7]	Defines an array names POSA with seven entries
DM SPEED[100]	Defines an array named speed with 100 entries
DM POSA[0]	Frees array space

Assignment of Array Entries

Like variables, each array element can be assigned a value. Assigned values can be numbers or returned values from instructions, functions and keywords.

Array elements are addressed starting at count 0. For example the first element in the POSA array (defined with the DM command, DM POSA[7]) would be specified as POSA[0].

Values are assigned to array entries using the equal sign. Assignments are made one element at a time by specifying the element number with the associated array name.

NOTE: Arrays must be defined using the command, DM, before assigning entry values.

DM SPEED[10]	Dimension Speed Array
SPEED[1]=7650.2	Assigns the first element of the array the value 7650.2
SPEED[1]=	Returns array element value
POSX[10]=_TPA	Assigns the 11th element the position of A.
CON[2]=@COS[POS]*2	Assigns the 3rd element of the array the cosine of POS * 2.
TIMER[1]=TIME	Assigns the 2nd element of the array TIME

Using a Variable to Address Array Elements

An array element number can also be a variable. This allows array entries to be assigned sequentially using a counter.

This example records 10 position values at a rate of one value per 10 msec. The values are stored in an array named POS. The variable, COUNT, is used to increment the array element counter. This example can also be executed with the automatic data capture feature described below.

Instruction	**Interpretation**
#A	Begin Program
COUNT=0;DM POS[10]	Initialize counter and define array
#LOOP	Begin loop
WT 10	Wait 10 msec
POS[COUNT]=_TPA	Record position into array element
POS[COUNT]=	Report position
COUNT=COUNT+1	Increment counter
JP #LOOP,COUNT<10	Loop until 10 elements have been stored
EN	End Program

Uploading and Downloading Arrays to On Board Memory

Arrays may be uploaded and downloaded using the QU and QD commands.

QU array[],start,end,delim

QD array[],start,end

where array is an array name such as A[].

Start is the first element of array (default=0)

End is the last element of array (default=last element)

Delim specifies whether the array data is separated by a comma (delim=1) or a carriage return (delim=0).

The file is terminated using <control>Z, <control>Q, <control>D or \.

Automatic Data Capture into Arrays

The DMC-3425 provides a special feature for automatic capture of data such as position, position error, inputs or torque. This is useful for teaching motion trajectories or observing system performance. Up to four types of data can be captured and stored in four arrays. The capture rate or time interval may be specified. Recording can be done as a one time event or as a circular continuous recording.

Command Summary - Automatic Data Capture

Command	Description
RA n[],m[],o[],p[]	Selects up to four arrays for data capture. The arrays must be defined with the DM command.
RD type1,type2,type3,type4	Selects the type of data to be recorded, where type1, type2, type3, and type 4 represent the various types of data (see table below). The order of data type is important and corresponds with the order of n,m,o,p arrays in the RA command.
RC n,m	The RC command begins data collection. Sets data capture time interval where n is an integer between 1 and 8 and designates 2^n msec between data. m is optional and specifies the number of elements to be captured. If m is not defined, the number of elements defaults to the smallest array defined by DM. When m is a negative number, the recording is done continuously in a circular manner. _RD is the recording pointer and indicates the address of the next array element. n=0 stops recording.
RC?	Returns a 0 or 1 where, 0 denotes not recording, 1 specifies recording in progress

Data Types for Recording:

Data Type	Description
_DEA	2nd encoder position (dual encoder)
_TPA	Encoder position
_TEA	Position error
_SHA	Commanded position
_RLA	Latched position
_TI	Inputs
_OP	Output
_TSA	Switches (only bit 0-4 valid)
_SCA	Stop code
_NOA	Status bits
_TTA	Torque (reports digital value +/-8097)

NOTE: B, C, D, E, F, G, or H may replace A for capturing data on other axes.

Operand Summary - Automatic Data Capture

_RC	Returns a 0 or 1 where, 0 denotes not recording, 1 specifies recording in progress
_RD	Returns address of next array element.

Example - Recording into An Array

Instruction	Interpretation
#RECORD	Begin program
DM APOS[300],BPOS[300]	Define A,B position arrays
DM AERR[300],BERR[300]	Define A,B error arrays
RA APOS[],AERR[],BPOS[],BERR[]	Select arrays for capture
RD _TPA,_TEA,_TPB,_TEB	Select data types
PR 10000,20000	Specify move distance

```
RC1                          Start recording now, at rate of 2 msec
BG AB                        Begin motion
#A;JP #A,RC=1                Loop until done
MG "DONE"                    Print message
EN                           End program
#PLAY                        Play back
N=0                          Initial Counter
JP# DONE,N>300               Exit if done
N=                           Print Counter
A POS[N]=                    Print A position
B POS[N]=                    Print B position
AERR[N]=                     Print A error
BERR[N]=                     Print B error
N=N+1                        Increment Counter
#DONE                        Done
EN                           End Program
```

Deallocating Array Space

Array space may be deallocated using the DA command followed by the array name. DA*[0] deallocates all the arrays.

Outputting Numbers and Strings

Numerical and string data can be output from the controller using several methods. The message command, MG, can output string and numerical data. Also, the controller can be commanded to return the values of variables and arrays, as well as other information using the interrogation commands (the interrogation commands are described in chapter 5).

Sending Messages

Messages may be sent to the bus using the message command, MG. This command sends specified text and numerical or string data from variables or arrays to the screen.

Text strings are specified in quotes and variable or array data is designated by the name of the variable or array. For example:

```
MG "The Final Value is", RESULT
```

In addition to variables, functions and commands, responses can be used in the message command. For example:

```
MG "Analog input is", @AN[1]
MG "The Position of A is", _TPA
```

Specifying the Port for Messages:

By default, messages will be sent through the port from which the data was requested. However, the port can be specified with the specifier, {P1} for the main serial port or {Ea} for the Ethernet handle. ‘a’ will be the handle letter, A through H.

```
MG {P1} "Hello World"        Sends message to Serial
```

Formatting Messages

String variables can be formatted using the specifier, {Sn} where n is the number of characters, 1 thru 6. For example:

```
MG STR {S3}
```

This statement returns 3 characters of the string variable named STR.

Numeric data may be formatted using the {Fn.m} expression following the completed MG statement. {$n.m} formats data in HEX instead of decimal. The actual numerical value will be formatted with n characters to the left of the decimal and m characters to the right of the decimal. Leading zeros will be used to display specified format.

For example::

```
MG "The Final Value is", RESULT {F5.2}
```

If the value of the variable RESULT is equal to 4.1, this statement returns the following:

The Final Value is 00004.10

If the value of the variable RESULT is equal to 999999.999, the above message statement returns the following:

The Final Value is 99999.99

The message command normally sends a carriage return and line feed following the statement. The carriage return and the line feed may be suppressed by sending {N} at the end of the statement. This is useful when a text string needs to surround a numeric value.

Example:

```
#A
JG 50000;BGA;ASA
MG "The Speed is", _TVA {F5.1} {N}
MG "counts/sec"
EN
```

When #A is executed, the above example will appear on the screen as:

```
The speed is 50000 counts/sec
```

Using the MG Command to Configure Terminals

The MG command can be used to configure a terminal. Any ASCII character can be sent by using the format {^n} where n is any integer between 1 and 255.

Example:

```
MG {^07} {^255}
```

sends the ASCII characters represented by 7 and 255 to the bus.

Summary of Message Functions

Function	Description
" "	Surrounds text string
{Fn.m}	Formats numeric values in decimal n digits to the right of the decimal point and m digits to the left
{P1}or {Ea}	Send message to Main Serial Port or Ethernet Port
{$n.m}	Formats numeric values in hexadecimal
{^n}	Sends ASCII character specified by integer n
{N}	Suppresses carriage return/line feed
{Sn}	Sends the first n characters of a string variable, where n is 1 thru 6.

Displaying Variables and Arrays

Variables and arrays may be sent to the screen using the format, variable= **or** array[x]=. For example, V1= , returns the value of V1.

Example - Printing a Variable and an Array element

Instruction	Interpretation
#DISPLAY	Label
DM POSA[7]	Define Array POSA with 7 entries
PR 1000	Position Command
BGA	Begin
AMA	After Motion
V1=_TPA	Assign Variable V1
POSA[1]=_TPA	Assign the first entry
V1=	Print V1

Interrogation Commands

The DMC-3425 has a set of commands that directly interrogate the controller. When these command are entered, the requested data is returned in decimal format on the next line followed by a carriage return and line feed. The format of the returned data can be changed using the Position Format (PF), and Leading Zeros (LZ) command. For a complete description of interrogation commands, see Ch 5.

Using the PF Command to Format Response from Interrogation Commands

The command, PF, can change format of the values returned by these interrogation commands:

BL ?	LE ?
DE ?	PA ?
DP ?	PR ?
EM ?	TN ?
FL ?	VE ?
IP ?	TE
TP	

The numeric values may be formatted in decimal or hexadecimal with a specified number of digits to the right and left of the decimal point using the PF command.

Position Format is specified by:

PF m.n

where m is the number of digits to the left of the decimal point (0 thru 10) and n is the number of digits to the right of the decimal point (0 thru 4) A negative sign for m specifies hexadecimal format.

Hex values are returned preceded by a $ and in 2's complement. Hex values should be input as signed 2's complement, where negative numbers have a negative sign. The default format is PF 10.0.

If the number of decimal places specified by PF is less than the actual value, a nine appears in all the decimal places.

Example

Instruction	**Interpretation**
:DP21	Define position
:TPA	Tell position
0000000021	Default format
:PF4	Change format to 4 places
:TPA	Tell position
0021	New format
:PF-4	Change to hexadecimal format
:TPA	Tell Position
$0015	Hexadecimal value
:PF2	Format 2 places
:TPA	Tell Position
99	Returns 99 if position greater than 99

Removing Leading Zeros from Response to Interrogation Commands

The leading zeros on data returned as a response to interrogation commands can be removed by the use of the command, LZ.

LZ0	Disables the LZ function
TP	Tell Position Interrogation Command
-0000000009, 0000000005	Response (With Leading Zeros)
LZ1	Enables the LZ function
TP	Tell Position Interrogation Command
-9, 5	Response (Without Leading Zeros)

Local Formatting of Response of Interrogation Commands

The response of interrogation commands may be formatted locally. To format locally, use the command, {Fn.m} or {$n.m} on the same line as the interrogation command. The symbol F specifies that the response should be returned in decimal format and $ specifies hexadecimal. n is the number of digits to the left of the decimal, and m is the number of digits to the right of the decimal.

TP {F2.2}	Tell Position in decimal format 2.2
-05.00, 05.00, 00.00, 07.00	Response from Interrogation Command
TP {$4.2}	Tell Position in hexadecimal format 4.2
FFFB.00,$0005.00,$0000.00,$0007.00	Response from Interrogation Command

Formatting Variables and Array Elements

The Variable Format (VF) command is used to format variables and array elements. The VF command is specified by:

VF m.n

where m is the number of digits to the left of the decimal point (0 thru 10) and n is the number of digits to the right of the decimal point (0 thru 4).

A negative sign for m specifies hexadecimal format. The default format for VF is VF 10.4

Hex values are returned preceded by a $ and in 2's complement.

Instruction	Interpretation
V1=10	Assign V1
V1=	Return V1
:0000000010.0000	Response - Default format
VF2.2	Change format
V1=	Return V1
:10.00	Response - New format
VF-2.2	Specify hex format
V1=	Return V1
$0A.00	Response - Hex value
VF1	Change format
V1=	Return V1
:9	Response - Overflow

Local Formatting of Variables

PF and VF commands are global format commands that affect the format of all relevant returned values and variables. Variables may also be formatted locally. To format locally, use the command, {Fn.m} or {$n.m} following the variable name and the '=' symbol. F specifies decimal and $ specifies hexadecimal. n is the number of digits to the left of the decimal, and m is the number of digits to the right of the decimal.

Instruction	Interpretation
V1=10	Assign V1
V1=	Return V1
:0000000010.0000	Default Format
V1={F4.2}	Specify local format
:0010.00	New format
V1={$4.2}	Specify hex format
:$000A.00	Hex value
V1="ALPHA"	Assign string "ALPHA" to V1
V1={S4}	Specify string format first 4 characters
:ALPH	

The local format is also used with the MG command.

Converting to User Units

Variables and arithmetic operations make it easy to input data in desired user units such as inches or RPM.

The DMC-3425 position parameters such as PR, PA and VP have units of quadrature counts. Speed parameters such as SP, JG and VS have units of counts/sec. Acceleration parameters such as AC, DC, VA and VD have units of counts/sec^2. The controller interprets time in milliseconds.

All input parameters must be converted into these units. For example, an operator can be prompted to input a number in revolutions. A program could be used such that the input number is converted into counts by multiplying it by the number of counts/revolution.

Instruction	Interpretation
#RUN	Label
IN "ENTER # OF REVOLUTIONS",N1	Prompt for revs
PR N1*2000	Convert to counts
IN "ENTER SPEED IN RPM",S1	Prompt for RPMs
SP S1*2000/60	Convert to counts/sec
IN "ENTER ACCEL IN RAD/SEC2",A1	Prompt for ACCEL
AC A1*2000/(2*3.14)	Convert to counts/sec2
BG	Begin motion
EN	End program

Hardware I/O

Digital Outputs

The DMC-3425 has 3 uncommitted outputs. Each bit on the output port may be set and cleared with the software instructions SB (Set Bit) and CB(Clear Bit), or OB (define output bit).

Example- Set Bit and Clear Bit

Instruction	Interpretation
SB3	Sets bit 3 of output port
CB2	Clears bit 2 of output port

Example- Output Bit

The Output Bit (OB) instruction is useful for setting or clearing outputs depending on the value of a variable, array, input or expression. Any non-zero value results in a set bit.

Instruction	Interpretation
OB1, POS	Set Output 1 if the variable POS is non-zero. Clear Output 1 if POS equals 0.
OB 2, @IN [1]	Set Output 2 if Input 1 is high. If Input 1 is low, clear Output 2.
OB 3, @IN [1]&@IN [2]	Set Output 3 only if Input 1 and Input 2 are high.
OB 3, COUNT [1]	Set Output 3 if element 1 in the array COUNT is non-zero.

The output port can be set by specifying an 8-bit word using the instruction OP (Output Port). This instruction allows a single command to define the state of the output port, where 2^0 is output 1, 2^1 is output 2 and so on. A 1 designates that the output is on.

Example- Output Port

Instruction	Interpretation
OP6	Sets outputs 2 and 3 of output port to high. All other bits are 0. ($2^1 + 2^2 = 6$)
OP0	Clears all bits of output port to zero
OP 255	Sets all bits of output port to one.

The output port is useful for setting relays or controlling external switches and events during a motion sequence.

Example - Turn on output after move

Instruction	Interpretation
#OUTPUT	Label
PR 2000	Position Command
BG	Begin
AM	After move
SB1	Set Output 1
WT 1000	Wait 1000 msec
CB1	Clear Output 1
EN	End

Digital Inputs

The general digital inputs for are accessed by using the @IN[n] function or the TI command. The @IN[n] function returns the logic level of the specified input, n.

Example - Using Inputs to control program flow

Instruction	Interpretation
JP #A,@IN[1]=0	Jump to A if input 1 is low
JP #B,@IN[2]=1	Jump to B if input 2 is high
AI 7	Wait until input 7 is high
AI -6	Wait until input 6 is low

Example - Start Motion on Switch

Motor A must turn at 4000 counts/sec when the user flips a panel switch to on. When panel switch is turned to off position, motor A must stop turning.

Solution: Connect panel switch to input 1 of DMC-3425. High on input 1 means switch is in on position.

Instruction	Interpretation
#S;JG 4000	Set speed
AI 1;BGA	Begin after input 1 goes high
AI -1;STA	Stop after input 1 goes low
AMA;JP #S	After motion, repeat
EN;	

Input Interrupt Function

The DMC-3425 provides an input interrupt function which causes the program to automatically execute the instructions following the #ININT label. This function is enabled using the II m,n,o command. The m specifies the beginning input and n specifies the final input in the range. The parameter o is an interrupt mask. If m and n are unused, o contains a number with the mask. A 1 designates that input to be enabled for an interrupt, where 2^0 is bit 1, 2^1 is bit 2 and so on. For example, II,,5 enables inputs 1 and 3 ($2^0 + 2^2 = 5$). The RI command (not EN) is used to return from the #ININT subroutine

A low input on any of the specified inputs will cause automatic execution of the #ININT subroutine. The Return from Interrupt (RI) command is used to return from this subroutine to the place in the program where the interrupt had occurred. If it is desired to return to somewhere else in the program after the execution of the #ININT subroutine, the Zero Stack (ZS) command is used followed by unconditional jump statements.

Example - Input Interrupt

Instruction	Interpretation
#A	Label #A
II 1	Enable input 1 for interrupt function
JG 30000,-20000	Set speeds on A and B axes
BG AB	Begin motion on A and B axes
#B	Label #B
TP AB	Report A and B axes positions
WT 1000	Wait 1000 milliseconds
JP #B	Jump to #B
EN	End of program
#ININT	Interrupt subroutine
MG "Interrupt has occurred"	Displays the message
ST AB	Stops motion on A and B axes
#LOOP;JP #LOOP,@IN[1]=0	Loop until Interrupt cleared
JG 15000,10000	Specify new speeds
WT 300	Wait 300 milliseconds
BG AB	Begin motion on A and B axes
RI	Return from Interrupt subroutine

Analog Inputs

The DMC-3425 provides two analog inputs. The value of these inputs in volts may be read using the @AN[n] function where n is the analog input 1 or 2. The resolution of the standard Analog-to-Digital conversion is 12 bits. Analog inputs are useful for reading special sensors such as temperature, tension or pressure.

The following examples show programs that cause the motor to follow an analog signal. The first example is a point-to-point move. The second example shows a continuous move.

Example - Position Follower (Point-to-Point)

Objective - The motor must follow an analog signal. When the analog signal varies by 10V, motor must move 10000 counts.

Method: Read the analog input and command A to move to that point.

Instruction	Interpretation
#Points	Label
SP 7000	Speed
AC 80000;DC 80000	Acceleration
#Loop	
VP=@AN[1]*1000	Read and analog input, compute position
PA VP	Command position
BGA	Start motion
AMA	After completion
JP #Loop	Repeat
EN	End

Example - Position Follower (Continuous Move)

Method: Read the analog input, compute the commanded position and the position error. Command the motor to run at a speed in proportions to the position error.

Instruction	Interpretation
#Cont	Label
AC 80000;DC 80000	Acceleration rate
JG 0	Start job mode
BGA	Start motion
#Loop	
VP=@AN[1]*1000	Compute desired position
VE=VP-_TPA	Find position error
VEL=VE*20	Compute velocity
JG VEL	Change velocity
JP #Loop	Change velocity
EN	End

Extended I/O of the DMC-3425 Controller

The DMC-3425 controller offers an option for 64 additional I/O, called the daughter board DB-14064. This I/O is known as extended I/O and can be configured as inputs or outputs in 8 bit increments through software. The I/O points are accessed through 2 50-pin high-density connectors.

Configuring the I/O of the DMC-3425

The extended I/O can be configured as outputs in blocks of 8. The I/O is configured as all Inputs by default. The extended I/O is denoted as blocks 2-9 or bits 17-80.

The command, CO, is used to configure the extended I/O as inputs or outputs. The CO command has one field:

CO n

where n is a decimal value which represents a binary number. Each bit of the binary number represents one block of extended I/O. When set to 1, the corresponding block is configured as an output.

NOTE: The CO command must be sent to slave controllers using the SA command.

The least significant bit represents block 2 and the most significant bit represents block 9. The decimal value can be calculated by the following formula. $n = n_2 + 2*n_3 + 4*n_4 + 8*n_5 +16* n_6 +32* n_7 +64* n_8 +128* n_9$ where n_x represents the block. If the n_x value is a one, then the block of 8 I/O points is to be configured as an output. If the n_x value is a zero, then the block of 8 I/O points will be configured as an input. For example, if block 4 and 5 is to be configured as an output, CO 12 is issued.

8-Bit I/O Block	Block	Binary Representation	Decimal Value for Block
17-24	2	2^0	1
25-32	3	$2^1 1$	2
33-40	4	2^2	4
41-48	5	2^3	8
49-56	6	2^4	16
57-64	7	2^5	32
65-72	8	2^6	64
73-80	9	2^7	128

The simplest method for determining n:

Step 1. Determine which 8-bit I/O blocks to be configured as outputs.

Step 2. From the table, determine the ***decimal value*** for each I/O block to be set as an output.

Step 3. Add up all of the values determined in step 2. This is the value to be used for n.

For example, if blocks 2 and 3 are to be outputs, then n is 3 and the command, CO3, should be issued. **NOTE:** This calculation is identical to the formula: $n = n_2 + 2*n_3 + 4*n_4 + 8*n_5 +16* n_6 +32* n_7 +64* n_8 +128* n_9$ where n_x represents the block.

Saving the State of the Outputs in Non-Volatile Memory

The configuration of the extended I/O and the state of the outputs can be stored in the EEPROM with the BN command. If no value has been set, the default of CO 0 is used (all blocks are inputs).

Accessing Extended I/O

When configured as an output, each I/O point may be defined with the SBn and CBn commands (where n=1 through 8 and 17 through 80). Outputs may also be defined with the conditional command, OBn (where n=1 through 8 and 17 through 80).

The command, OP, may also be used to set output bits, specified as blocks of data. The OP command accepts 5 parameters. The first parameter sets the values of the main output port of the controller (Outputs 1-8, block 0). The additional parameters set the value of the extended I/O as outlined:

OP m,a,b,c,d

where m is the decimal representation of the bits 1-8 (values from 0 to 255) and a,b,c,d represent the extended I/O in consecutive groups of 16 bits. (values from 0 to 65535). Arguments given for I/O points that are configured as inputs will be ignored. The following table describes the arguments used to set the state of outputs.

Argument	Blocks	Bits	Description
m	0	1-8	General Outputs
a	2,3	17-32	Extended I/O
b	4,5	33-48	Extended I/O
c	6,7	49-64	Extended I/O
d	8,9	65-80	Extended I/O

For example, if block 8 is configured as an output, the following command may be issued:

OP 7,,,,7

This command will set bits 1,2,3 (block 0) and bits 65,66,67 (block 8) to 1. Bits 4 through 8 and bits 68 through 80 will be set to 0. All other bits are unaffected.

When accessing I/O blocks configured as inputs, use the TIn command. The argument 'n' refers to the block to be read (n=0,2,3,4,5,6,7,8 or 9). The value returned will be a decimal representation of the corresponding bits.

Individual bits can be queried using the @IN[n] function (where n=1 through 8 or 17 through 80). If the following command is issued;

MG @IN[17]

the controller will return the state of the least significant bit of block 2 (assuming block 2 is configured as an input).

Interfacing to Grayhill or OPTO-22 G4PB24

The DMC-3425 2 50 Pin IDC connectors which are compatible with I/O mounting racks such as Grayhill 70GRCM32-HL and OPTO-22 G4PB24. The 50 pin ribbon cables can connect directly into the I/O mounting racks.

When using the OPTO-22 G4PB24 I/O mounting rack, the user will only have access to 48 of the 64 I/O points available on the controller. Block 5 and Block 9 must be configured as inputs and will be grounded by the I/O rack.

Example Applications

Wire Cutter

An operator activates a start switch. This causes a motor to advance the wire a distance of 10". When the motion stops, the controller generates an output signal that activates the cutter. Allowing 100 ms for the cutting completes the cycle.

Suppose that the motor drives the wire by a roller with a 2" diameter. Also assume that the encoder resolution is 1000 lines per revolution. Since the circumference of the roller equals 2π inches, and it corresponds to 4000 quadrature, one inch of travel equals:

$4000/2\pi = 637$ count/inch

This implies that a distance of 10 inches equals 6370 counts, and a slew speed of 5 inches per second, for example, equals 3185 count/sec.

The input signal may be applied to I1, for example, and the output signal is chosen as output 1. The motor velocity profile and the related input and output signals are shown in Fig. 7.1.

The program starts at a state that we define as #A. Here the controller waits for the input pulse on I1. As soon as the pulse is given, the controller starts the forward motion.

Upon completion of the forward move, the controller outputs a pulse for 20 ms and then waits an additional 80 ms before returning to #A for a new cycle.

Instruction	Interpretation
#A	Label
AI1	Wait for input 1
PR 6370	Distance
SP 3185	Speed
BGA	Start Motion
AMA	After motion is complete
SB1	Set output bit 1
WT 20	Wait 20 ms
CB1	Clear output bit 1
WT 80	Wait 80 ms
JP #A	Repeat the process

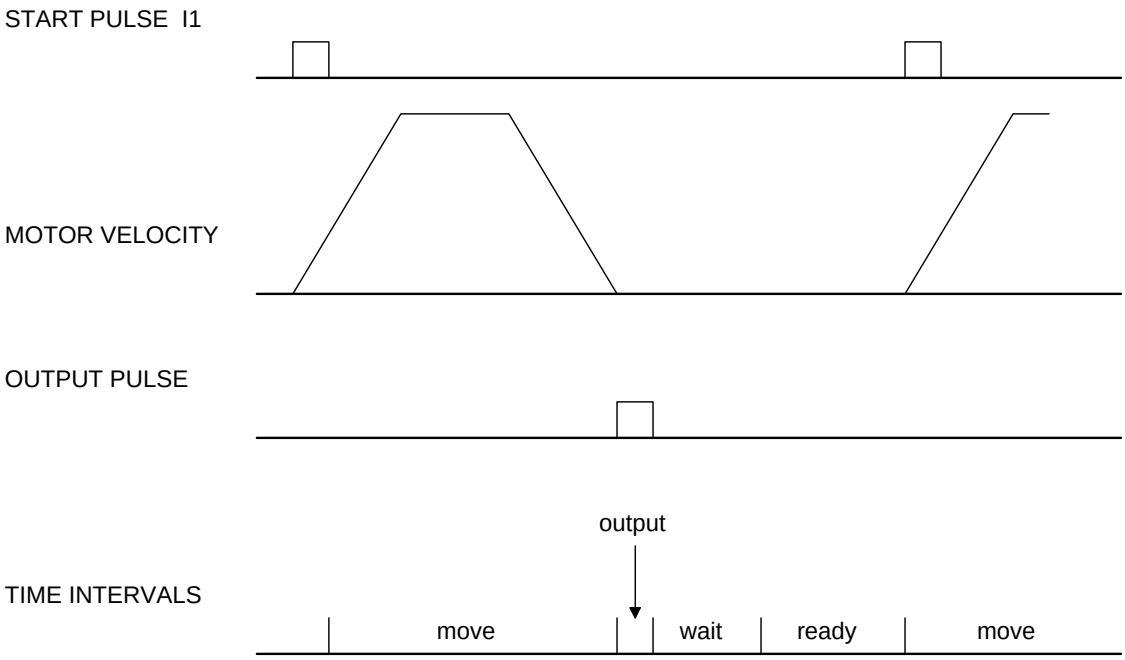


Figure 7.1 - Motor Velocity and the Associated Input/Output signals

A-B (X-Y) Table Controller

An A-B-C system must cut the pattern shown in Fig. 7.2. The A-B table moves the plate while the C-axis raises and lowers the cutting tool.

The solid curves in Fig. 7.2 indicate sections where cutting takes place. Those must be performed at a feedrate of 1 inch per second. The dashed line corresponds to non-cutting moves and should be performed at 5 inch per second. The acceleration rate is 0.1 g.

The motion starts at point A, with the C-axis raised. An A-B motion to point B is followed by lowering the C-axis and performing a cut along the circle. Once the circular motion is completed, the C-axis is raised and the motion continues to point C, etc.

Assume that all of the 3 axes are driven by lead screws with 10 turns-per-inch pitch. Also assume encoder resolution of 1000 lines per revolution. This results in the relationship:

1 inch = 40,000 counts

and the speeds of

1 in/sec = 40,000 count/sec

5 in/sec = 200,000 count/sec

an acceleration rate of 0.1g equals

$0.1g = 38.6 \text{ in/s2} = 1{,}544{,}000 \text{ count/s}^2$

Note that the circular path has a radius of 2" or 80000 counts, and the motion starts at the angle of 270° and traverses 360° in the CW (negative direction). Such a path is specified with the instruction

CR 80000,270,-360

Further assume that the C must move 2" at a linear speed of 2" per second. The required motion is performed by the following instructions:

Instruction	**Interpretation**
#A	Label
VM AB	Circular interpolation for AB
VP 160000,160000	Positions
VE	End Vector Motion
VS 200000	Vector Speed
VA 1544000	Vector Acceleration
BGS	Start Motion
AMS	When motion is complete
PR,,-80000	Move C down
SP,,80000	C speed
BGC	Start C motion
AMC	Wait for completion of C motion
CR 80000,270,-360	Circle
VE	
VS 40000	Feedrate
BGS	Start circular move
AMS	Wait for completion
PR,,80000	Move C up
BGC	Start C move
AMC	Wait for C completion
PR –21600	Move A
SP 20000	Speed A
BGA	Start A
AMA	Wait for A completion
PR,,-80000	Lower C
BGC	
AMC	
CR 80000,270,-360	C second circle move
VE	
VS 40000	

```
BGS
AMS
PR,,80000              Raise C
BGC
AMC
VP -37600,-16000       Return AB to start
VE
VS 200000
BGS
AMS
EN
```

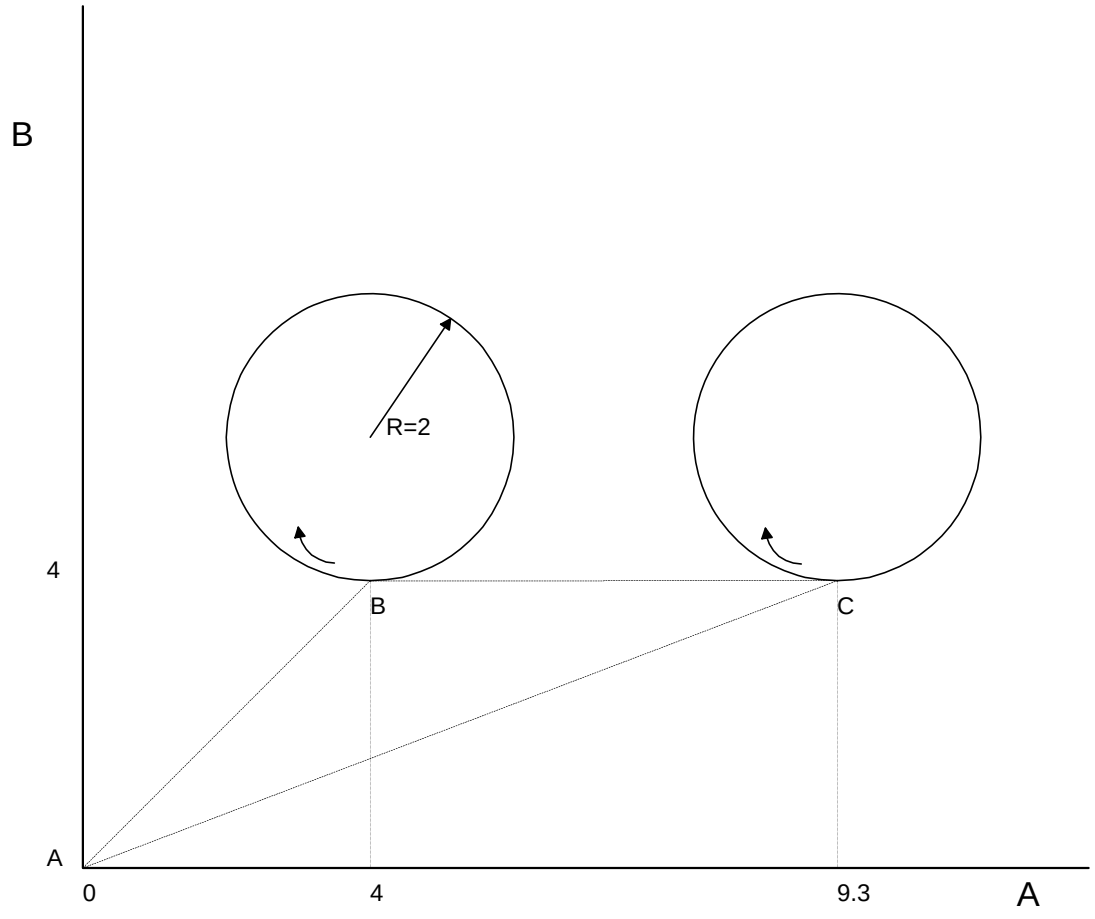


Figure 7.2 - Motor Velocity and the Associated Input/Output signals

Speed Control by Joystick

The speed of a motor is controlled by a joystick. The joystick produces a signal in the range between -10V and +10V. The objective is to drive the motor at a speed proportional to the input voltage.

Assume that a full voltage of 10 Volts must produce a motor speed of 3000 rpm with an encoder resolution of 1000 lines or 4000 count/rev. This speed equals:

3000 rpm = 50 rev/sec = 200000 count/sec

The program reads the input voltage periodically and assigns its value to the variable VIN. To get a speed of 200,000 ct/sec for 10 volts, we select the speed as

Speed = 20000 x VIN

The corresponding velocity for the motor is assigned to the VEL variable.

Instruction

```
#A
JG0
BGA
#B
VIN=@AN[1]
VEL=VIN*20000
JG VEL
JP #B
EN
```

Position Control by Joystick

This system requires the position of the motor to be proportional to the joystick angle. Furthermore, the ratio between the two positions must be programmable. For example, if the control ratio is 5:1, it implies that when the joystick voltage is 5 Volts, corresponding to 1028 counts, the required motor position must be 5120 counts. The variable V3 changes the position ratio.

Instruction	**Interpretation**
#A	Label
V3=5	Initial position ratio
DP0	Define the starting position
JG0	Set motor in jog mode as zero
BGA	Start
#B	
V1=@AN[1]	Read analog input
V2=V1*V3	Compute the desired position
V4=V2-_TPA-_TEA	Find the following error
V5=V4*20	Compute a proportional speed
JG V5	Change the speed
JP #B	Repeat the process
EN	End

THIS PAGE LEFT BLANK INTENTIONALLY

Chapter 8 Hardware & Software Protection

Introduction

The DMC-3425 provides several hardware and software features to check for error conditions and to inhibit the motor on error. These features help protect the system components from damage.

WARNING: Machinery in motion can be dangerous! It is the responsibility of the user to design effective error handling and safety protection as part of the machine. Since the DMC-3425 is an integral part of the machine, the engineer should design his overall system with protection against a possible component failure on the DMC-3425. Galil shall not be liable or responsible for any incidental or consequential damages.

Hardware Protection

The DMC-3425 includes hardware input and output protection lines for error and mechanical limit conditions. These include:

Output Protection Lines

Amp Enable - This signal goes low when the motor off command is given, when the position error exceeds the value specified by the Error Limit (ER) command, or when an off-on-error condition is enabled (OE1) and the abort command is given. Each axis amplifier has a separate enable line. This signal also goes low when the watch-dog timer is activated. *Note: The standard configuration of the AEN signal is TTL active low. Both the polarity and the amplitude can be changed if you are using the ICM-1460 interface board. To make these changes, see section entitled 'Amplifier Interface'.*

Note: There is only one amplifier enable signal for the DMC-3425. Therefore, both amplifiers will be controlled by the same enable output.

Error Output - The error output is a TTL signal that indicates an error condition in the controller. This signal is available on the interconnect module as ERROR. When the error signal is low, this indicates one of the following error conditions:

1. At least one axis has a position error greater than the error limit. The error limit is set by using the command ER.
2. The reset line on the controller is held low or is being affected by noise.
3. There is a failure on the controller and the processor is resetting itself.

4. There is a failure with the output IC that drives the error signal.

Input Protection Lines

Abort - A low input stops commanded motion instantly without a controlled deceleration. For any axis in which the Off-On-Error function is enabled, the amplifiers will be disabled. This could cause the motor to 'coast' to a stop. If the Off-On-Error function is not enabled, the motor will instantaneously stop and servo at the current position. The Off-On-Error function is further discussed in this chapter.

Forward Limit Switch - Low input inhibits motion in forward direction. (The CN command can be used to change the polarity of the limit switches.) If the motor is moving in the forward direction when the limit switch is activated, the motion will decelerate and stop. In addition, if the motor is moving in the forward direction, the controller will automatically jump to the limit switch subroutine, #LIMSWI (if such a routine has been written by the user).

Reverse Limit Switch - Low input inhibits motion in reverse direction. (The CN command can be used to change the polarity of the limit switches.) If the motor is moving in the reverse direction when the limit switch is activated, the motion will decelerate and stop. In addition, if the motor is moving in the reverse direction, the controller will automatically jump to the limit switch subroutine, #LIMSWI (if such a routine has been written by the user).

Software Protection

The DMC-3425 provides a programmable error limit. The error limit refers to a difference in the actual and commanded position of the motor. This limit can be set for any number between 1 and 32767 using the ER n command. The default value for ER is 16384.

Example:

ER 200,300	Set A-axis error limit for 200, B-axis error limit to 300
ER,1	Set B-axis error limit to 1 count.

The units of the error limit are quadrature counts. The error is the difference between the command position and actual encoder position. If the absolute value of the error exceeds the value specified by ER, the DMC-3425 will generate signals to warn the host system of the error condition. These signals include:

Signal or Function	State if Error Occurs
# POSERR	Jumps to automatic excess position error subroutine (if included in program)
Error Light	Turns on
OE Function	Shuts motor off if OE1
AEN Output Line	Goes low

The Jump if Condition statement is useful for branching within the program due to an error. The position error of A and B can be monitored during execution using the TE command.

Programmable Position Limits

The DMC-3425 provides programmable forward and reverse position limits. These are set by the BL (Backwards Limit) and FL (Forward Limit) software commands. Once a position limit is specified, the DMC-3425 will not accept position commands beyond the limit. Motion beyond the limit is also prevented.

Example:

DP0,0,	Define Position
BL -2000,-4000	Set Reverse position limit
FL 2000,4000	Set Forward position limit
JG 2000,2000	Jog
BG AB	Begin

Execution of the above example will cause the motor to slew at the given jog speed until the forward position limit is reached. Motion will stop once the limit is hit.

Off-On-Error

The DMC-3425 controller has a built in function that can turn off the motors under certain error conditions. This function is know as 'Off-On-Error". To activate the OE function for each axis, specify 1 for A and B axes. To disable this function, specify 0 for the axes. When the function is enabled, the corresponding motor will be disabled under the following 3 conditions:

1. The position error for the specified axis exceeds the limit set with the command, ER
2. The abort command is given
3. The abort input is activated with a low signal.

Note: If the motors are disabled while they are moving, they may 'coast' to a stop because they are no longer under servo control.

To re-enable the system, use the Servo Here (SH) command. The SH command will clear any position error and reset the commanded position to the actual position.

Examples:

OE 1,1	Enable off-on-error for A and B
OE 0,1	Enable off-on-error for B axis and disable off-on-error for A axis

Automatic Error Routine

The #POSERR label causes the statements following to be automatically executed if the error on any axis exceeds the error limit specified by ER. The error routine should be closed with the RE command. RE will cause the main program to be resumed where left off.

NOTE: The Error Subroutine will be entered again unless the error condition is gone.

Example:

Instruction	**Interpretation**
#A;JP #A;EN	"Dummy" program
#POSERR	Start error routine on error
MG "error"	Send message
SB 1	Fire relay
STA	Stop motor
AMA	After motor stops
SHA	Servo motor here to clear error
RE	Return to main program

NOTE: *An applications program must be executing for the #POSERR routine to function.*

Limit Switch Routine

The DMC-3425 provides forward and reverse limit switches that inhibit motion in the respective direction. There is also a special label for automatic execution of a limit switch subroutine. The #LIMSWI label specifies the start of the limit switch subroutine. This label causes the statements following to be automatically executed if any limit switch is activated. The RE command ends the subroutine and resumes the main program where it left off.

The state of the forward and reverse limit switches may also be interrogated or used in a conditional statement. The _LR condition specifies the reverse limit and _LF specifies the forward limit. A or B following _LR or _LF specifies the axis. The CN command can be used to configure the polarity of the limit switches.

Limit Switch Example:

Instruction	**Interpretation**
#A;JP #A;EN	Dummy Program
#LIMSWI	Limit Switch Utility
V1=_LFA	Check state of forward limit
V2=_LRA	Check state of reverse limit
JP#LF,V1=0	Jump to #LF if forward limit = low
JP#LR,V2=0	Jump to #LR if reverse limit = low
JP#END	Jump to end
#LF	#LF
MG "FORWARD LIMIT"	Send message
STA;AMA	Stop motion
PR-1000;BGA;AMA	Move in reverse
JP#END	End
#LR	#LR
MG "REVERSE LIMIT"	Send message
STA;AMA	Stop motion
PR1000;BGA;AMA	Move forward
#END	End
RE	Return to main program

NOTE: An applications program must be executing for #LIMSWI to function.

Chapter 9 Troubleshooting

Overview

The following discussion may help you get your system to work.

Potential problems have been divided into groups as follows:

1. Installation
2. Communication
3. Stability and Compensation
4. Operation

The various symptoms along with the cause and the remedy are described in the following tables.

Installation

Symptom	Cause	Remedy
Motor runs away when connected to amplifier with no additional inputs.	Amplifier offset too large.	Adjust amplifier offset
Same as above, but offset adjustment does not stop the motor.	Damaged amplifier.	Replace amplifier.
Same as above, but offset adjustment does not stop the motor.	Damaged amplifier.	Replace amplifier.
Controller does not read changes in encoder position.	Wrong encoder connections.	Check encoder wiring.
Same as above	Bad encoder	Check the encoder signals. Replace encoder if necessary.
Same as above	Bad controller	Connect the encoder to different axis input. If it works, controller failure. Repair or replace.

Communication

Symptom	Cause	Remedy
Using terminal emulator, cannot communicate with controller.	Selected comport incorrect	Try another comport
Same as above	Selected baud rate incorrect	Check to be sure that baud rate same as dip switch settings on controller, change as necessary.

Stability

Symptom	Cause	Remedy
Motor runs away when the loop is closed.	Wrong feedback polarity.	Invert the polarity of the loop by inverting the motor leads (brush type) or the encoder.
Motor oscillates.	Too high gain or too little damping.	Decrease KI and KP. Increase KD.

Operation

Symptom	Cause	Remedy
Controller rejects command. Responded with a ?	Anything.	Interrogate the cause with TC or TC1.
Motor does not complete move.	Noise on limit switches stops the motor.	To verify cause, check the stop code (SC). If caused by limit switch noise, reduce noise.
During a periodic operation, motor drifts slowly.	Encoder noise	Interrogate the position periodically. If controller states that the position is the same at different locations it implies encoder noise. Reduce noise. Use differential encoder inputs.
Same as above.	Programming error.	Avoid resetting position error at end of move with SH command.

Chapter 10 Theory of Operation

Overview

The following discussion covers the operation of motion control systems. A typical motion control system consists of the elements shown in Fig 10.1.

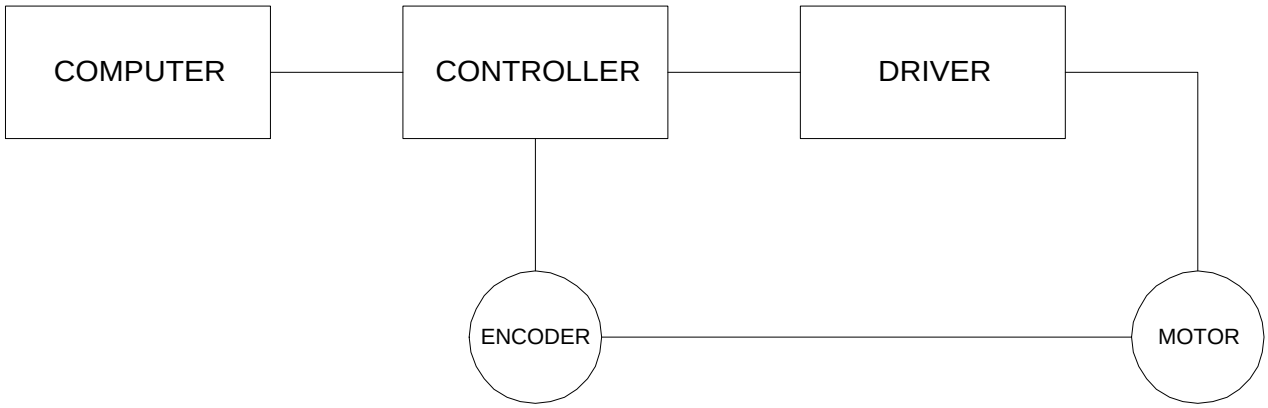


Figure 10.1 - Elements of Servo Systems

The operation of such a system can be divided into three levels, as illustrated in Fig. 10.2. The levels are:

1. Closing the Loop
2. Motion Profiling
3. Motion Programming

The first level, the closing of the loop, assures that the motor follows the commanded position. This is done by closing the position loop using a sensor. The operation at the basic level of closing the loop involves the subjects of modeling, analysis, and design. These subjects will be covered in the following discussions.

The motion profiling is the generation of the desired position function. This function, R(t), describes where the motor should be at every sampling period. Note that the profiling and the closing of the loop are independent functions. The profiling function determines where the motor should be and the closing of the loop forces the motor to follow the commanded position

The highest level of control is the motion program. This can be stored in the host computer or in the controller. This program describes the tasks in terms of the motors that need to be controlled, the distances and the speed.

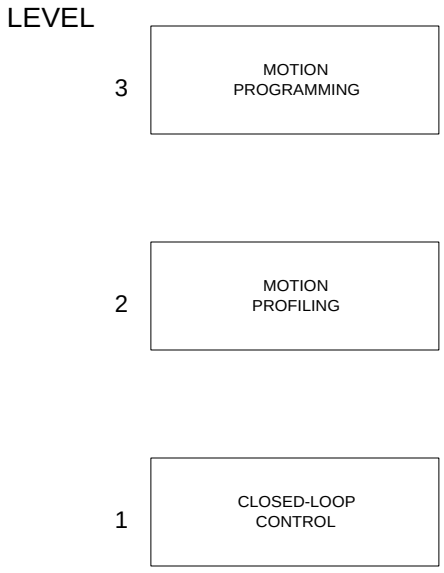


Figure 10.2 - Levels of Control Functions

The three levels of control may be viewed as different levels of management. The top manager, the motion program, may specify the following instruction, for example.

```
PR 6000,4000
SP 20000,20000
AC 200000,00000
BG A
AD 2000
BG B
EN
```

This program corresponds to the velocity profiles shown in Fig. 10.3. Note that the profiled positions show where the motors must be at any instant of time.

Finally, it remains up to the servo system to verify that the motor follows the profiled position by closing the servo loop.

The following section explains the operation of the servo system. First, it is explained qualitatively, and then the explanation is repeated using analytical tools for those who are more theoretically inclined.

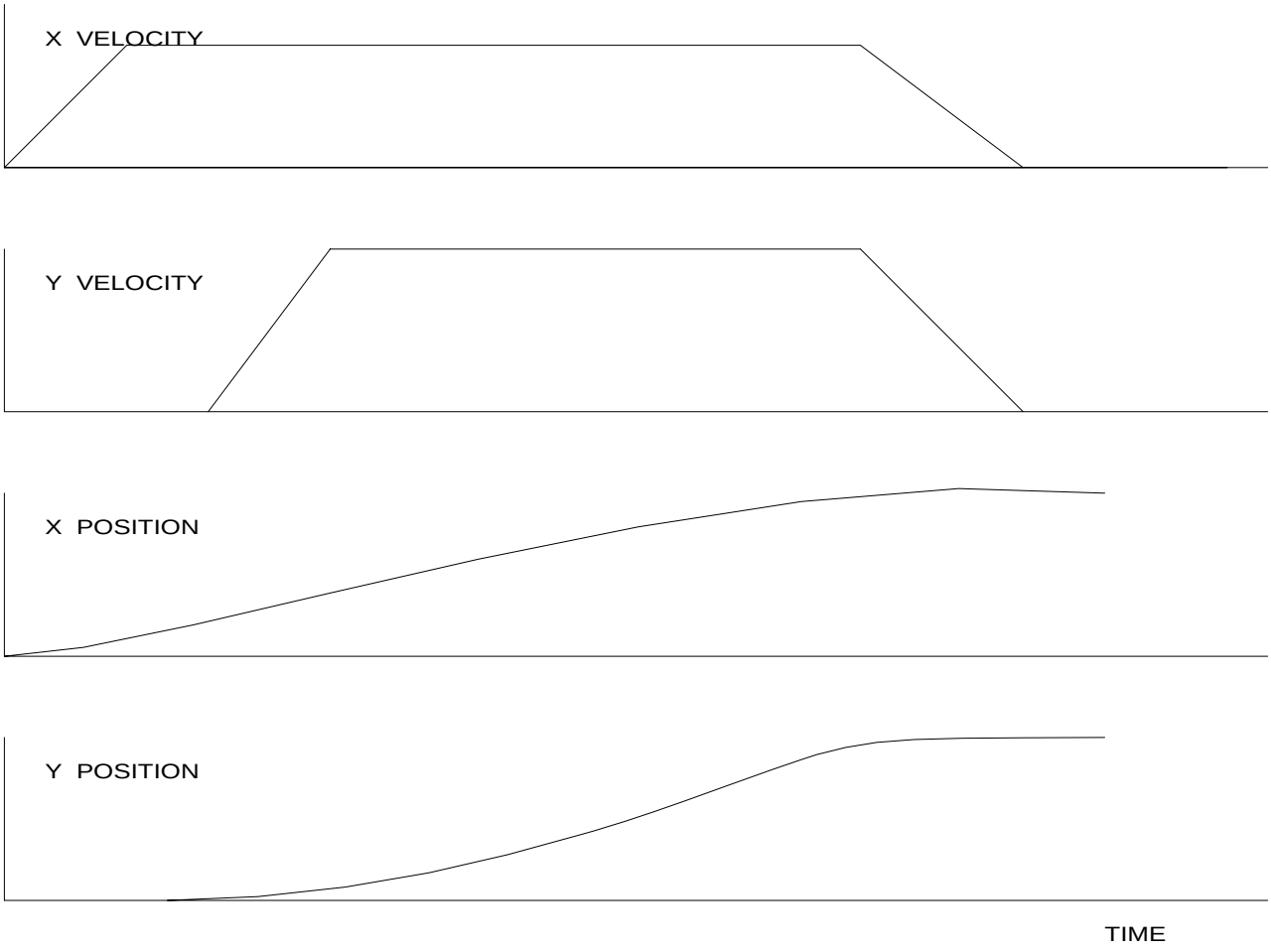


Figure 10.3 - Velocity and Position Profiles

Operation of Closed-Loop Systems

To understand the operation of a servo system, we may compare it to a familiar closed-loop operation, adjusting the water temperature in the shower. One control objective is to keep the temperature at a comfortable level, say 90 degrees F. To achieve that, our skin serves as a temperature sensor and reports to the brain (controller). The brain compares the actual temperature, which is called the feedback signal, with the desired level of 90 degrees F. The difference between the two levels is called the error signal. If the feedback temperature is too low, the error is positive, and it triggers an action which raises the water temperature until the temperature error is reduced sufficiently.

The closing of the servo loop is very similar. Suppose that we want the motor position to be at 90 degrees. The motor position is measured by a position sensor, often an encoder, and the position feedback is sent to the controller. Like the brain, the controller determines the position error, which is the difference between the commanded position of 90 degrees and the position feedback. The controller then outputs a signal that is proportional to the position error. This signal produces a proportional current in the motor, which causes a motion until the error is reduced. Once the error becomes small, the resulting current will be too small to overcome the friction, causing the motor to stop.

The analogy between adjusting the water temperature and closing the position loop carries further. We have all learned the hard way, that the hot water faucet should be turned at the "right" rate. If you turn

it too slowly, the temperature response will be slow, causing discomfort. Such a slow reaction is called over damped response.

The results may be worse if we turn the faucet too fast. The overreaction results in temperature oscillations. When the response of the system oscillates, we say that the system is unstable. Clearly, unstable responses are bad when we want a constant level.

What causes the oscillations? The basic cause for the instability is a combination of delayed reaction and high gain. In the case of the temperature control, the delay is due to the water flowing in the pipes. When the human reaction is too strong, the response becomes unstable.

Servo systems also become unstable if their gain is too high. The delay in servo systems is between the application of the current and its effect on the position. Note that the current must be applied long enough to cause a significant effect on the velocity, and the velocity change must last long enough to cause a position change. This delay, when coupled with high gain, causes instability.

This motion controller includes a special filter that is designed to help the stability and accuracy. Typically, such a filter produces, in addition to the proportional gain, damping and integrator. The combination of the three functions is referred to as a PID filter.

The filter parameters are represented by the three constants KP, KI and KD, which correspond to the proportional, integral and derivative term respectively.

The damping element of the filter acts as a predictor, thereby reducing the delay associated with the motor response.

The integrator function, represented by the parameter KI, improves the system accuracy. With the KI parameter, the motor does not stop until it reaches the desired position exactly, regardless of the level of friction or opposing torque.

The integrator also reduces the system stability. Therefore, it can be used only when the loop is stable and has a high gain.

The output of the filter is applied to a digital-to-analog converter (DAC). The resulting output signal in the range between +10 and -10 Volts is then applied to the amplifier and the motor.

The motor position, whether rotary or linear is measured by a sensor. The resulting signal, called position feedback, is returned to the controller for closing the loop.

The following section describes the operation in a detailed mathematical form, including modeling, analysis and design.

System Modeling

The elements of a servo system include the motor, driver, encoder and the controller. These elements are shown in Fig. 10.4. The mathematical model of the various components is given below.

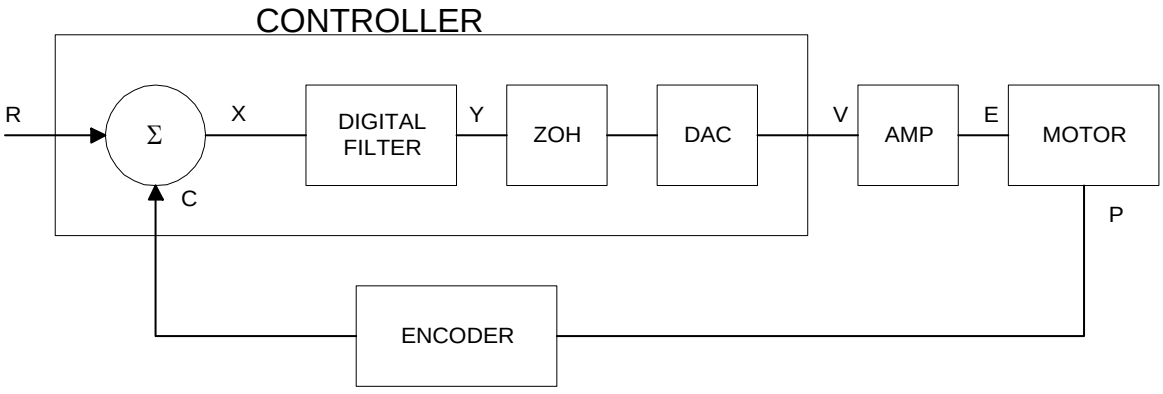


Figure 10.4 - Functional Elements of a Motion Control System

Motor-Amplifier

The motor amplifier may be configured in three modes:

1. Voltage Drive
2. Current Drive
3. Velocity Loop

The operation and modeling in the three modes is as follows:

Voltage Drive

The amplifier is a voltage source with a gain of Kv [V/V]. The transfer function relating the input voltage, V, to the motor position, P, is

$$P/V = K_V / \left[K_t S (ST_m + 1)(ST_e + 1) \right]$$

where

$$T_m = RJ / K_t^2 \quad [s]$$

and

$$T_e = L/R \quad [s]$$

and the motor parameters and units are

K_t	Torque constant [Nm/A]
R	Armature Resistance Ω
J	Combined inertia of motor and load [kg.m^2]
L	Armature Inductance [H]

When the motor parameters are given in English units, it is necessary to convert the quantities to MKS units. For example, consider a motor with the parameters:

K_t = 14.16 oz - in/A = 0.1 Nm/A

R = 2 Ω

J = 0.0283 oz-in-s^2 = 2.10^{-4} kg . m^2

L = 0.004H

Then the corresponding time constants are

T_m = 0.04 sec

and

T_e = 0.002 sec

Assuming that the amplifier gain is Kv = 4, the resulting transfer function is

P/V = 40/[s(0.04s+1)(0.002s+1)]

Current Drive

The current drive generates a current I, which is proportional to the input voltage, V, with a gain of Ka. The resulting transfer function in this case is

P/V = K_a K_t / Js^2

where Kt and J are as defined previously. For example, a current amplifier with K_a = 2 A/V with the motor described by the previous example will have the transfer function:

$$P/V = 1000/s^2 \qquad [rad/V]$$

If the motor is a DC brushless motor, it is driven by an amplifier that performs the commutation. The combined transfer function of motor amplifier combination is the same as that of a similar brush motor, as described by the previous equations.

Velocity Loop

The motor driver system may include a velocity loop where the motor velocity is sensed by a tachometer and is fed back to the amplifier. Such a system is illustrated in Fig. 10.5. Note that the transfer function between the input voltage V and the velocity ω is:

$$\omega /V = [K_a K_t/Js]/[1+K_a K_t K_g/Js] = 1/[K_g(sT_1+1)]$$

where the velocity time constant, T1, equals

$$T1 = J/K_a K_t K_g$$

This leads to the transfer function

$$P/V = 1/[K_g s(sT1+1)]$$

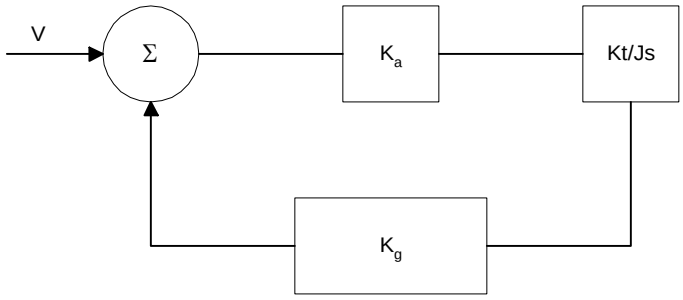
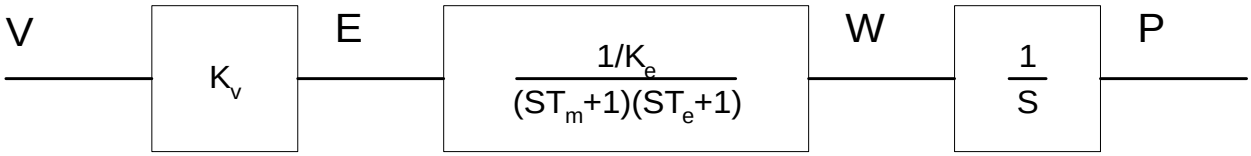


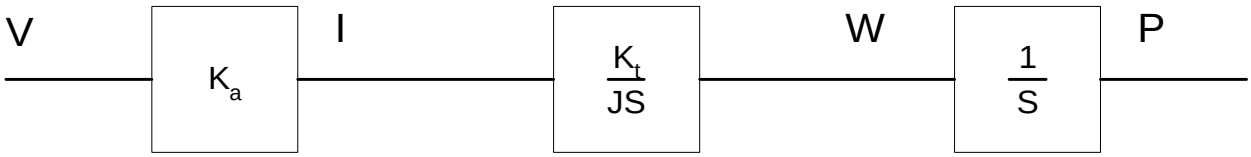
Figure 10.5 - Elements of velocity loops

The resulting functions derived above are illustrated by the block diagram of Fig. 10.6.

VOLTAGE SOURCE



CURRENT SOURCE



VELOCITY LOOP

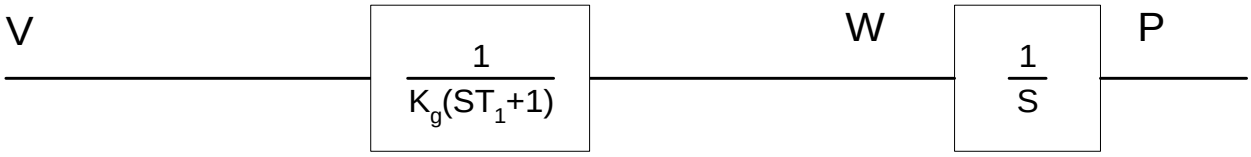


Figure 10.6 - Mathematical model of the motor and amplifier in three operational modes

Encoder

The encoder generates N pulses per revolution. It outputs two signals, Channel A and B, which are in quadrature. Due to the quadrature relationship between the encoder channels, the position resolution is increased to 4N quadrature counts/rev.

The model of the encoder can be represented by a gain of

$K_f = 4N/2\pi$ [count/rad]

For example, a 1000 lines/rev encoder is modeled as

$K_f = 638$

DAC

The DAC or D-to-A converter converts a 16-bit number to an analog voltage. The input range of the numbers is 65536 and the output voltage range is +/-10V or 20V. Therefore, the effective gain of the DAC is

K= 20/65536 = 0.0003 [V/count]

Digital Filter

The digital filter has three elements in series: PID, low-pass and a notch filter. The transfer functions of the filter elements are:

PID $D(z) = \frac{K(Z-A)}{Z} + \frac{CZ}{Z-1}$

Low-pass $L(z) = \frac{1-B}{Z-B}$

Notch $N(z) = \frac{(Z-z)(Z-\bar{z})}{(Z-p)(Z-\bar{p})}$

The filter parameters, K, A, C and B are selected by the instructions KP, KD, KI and PL, respectively. The relationship between the filter coefficients and the instructions are:

K = (KP + KD) • 4

A = KD/(KP + KD)

C = KI/2

B = PL

The PID and low-pass elements are equivalent to the continuous transfer function G(s).

G(s) = (P + sD + I/s) ∗ a/(S+a)

P = 4KP

D = 4T • KD

I = KI/2T

a = 1/T ln = (1/B)

where T is the sampling period.

For example, if the filter parameters of the DMC-2x00 are

KP = 4

KD = 36

KI = 2

PL = 0.75

T = 0.001 s

the digital filter coefficients are

K = 160

A = 0.9

C = 1

a = 250 rad/s

and the equivalent continuous filter, G(s), is

$$G(s) = [16 + 0.144s + 1000/s\} * 250/ (s+250)$$

The notch filter has two complex zeros, Z and z, and two complex poles, P and p.

The effect of the notch filter is to cancel the resonance affect by placing the complex zeros on top of the resonance poles. The notch poles, P and p, are programmable and are selected to have sufficient damping. It is best to select the notch parameters by the frequency terms. The poles and zeros have a frequency in Hz, selected by the command NF. The real part of the poles is set by NB and the real part of the zeros is set by NZ.

The simplest procedure for setting the notch filter is to identify the resonance frequency and set NF to the same value. Set NB to about one half of NF and set NZ to a low value between zero and 5.

ZOH

The ZOH, or zero-order-hold, represents the effect of the sampling process, where the motor command is updated once per sampling period. The effect of the ZOH can be modeled by the transfer function

$$H(s) = 1/(1+sT/2)$$

If the sampling period is T = 0.001, for example, H(s) becomes:

$$H(s) = 2000/(s+2000)$$

However, in most applications, H(s) may be approximated as one.

This completes the modeling of the system elements. Next, we discuss the system analysis.

System Analysis

To analyze the system, we start with a block diagram model of the system elements. The analysis procedure is illustrated in terms of the following example.

Consider a position control system with the DMC-2x00 controller and the following parameters:

$K_t = 0.1$	Nm/A	Torque constant
$J = 2.10^{-4}$	$kg.m^2$	System moment of inertia
R = 2	Ω	Motor resistance
$K_a = 4$	Amp/Volt	Current amplifier gain
KP = 12.5		Digital filter gain
KD = 245		Digital filter zero
KI = 0		No integrator
N = 500	Counts/rev	Encoder line density
T = 1	ms	Sample period

The transfer functions of the system elements are:

Motor

$M(s) = P/I = Kt/Js2 = 500/s^2$ [rad/A]

Amp

$K_a = 4$ [Amp/V]

DAC

$K_d = 0.0003$ [V/count]

Encoder

$K_f = 4N/2\pi = 318$ [count/rad]

ZOH

$2000/(s+2000)$

Digital Filter

KP = 12.5, KD = 245, T = 0.001

Therefore,

$D(z) = 1030\ (z-0.95)/Z$

Accordingly, the coefficients of the continuous filter are:

P = 50

D = 0.98

The filter equation may be written in the continuous equivalent form:

$G(s) = 50 + 0.98s = .098\ (s+51)$

The system elements are shown in Fig. 10.7.

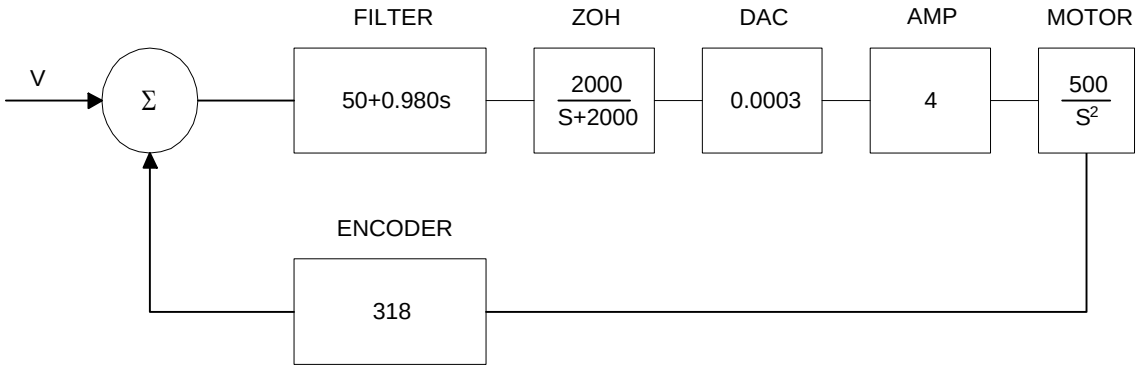


Figure 10.7 - Mathematical model of the control system

The open loop transfer function, A(s), is the product of all the elements in the loop.

$A = 390{,}000\ (s+51)/[s^2(s+2000)]$

To analyze the system stability, determine the crossover frequency, ω_c at which $A(j\ \omega_c)$ equals one. This can be done by the Bode plot of $A(j\ \omega_c)$, as shown in Fig. 10.8.

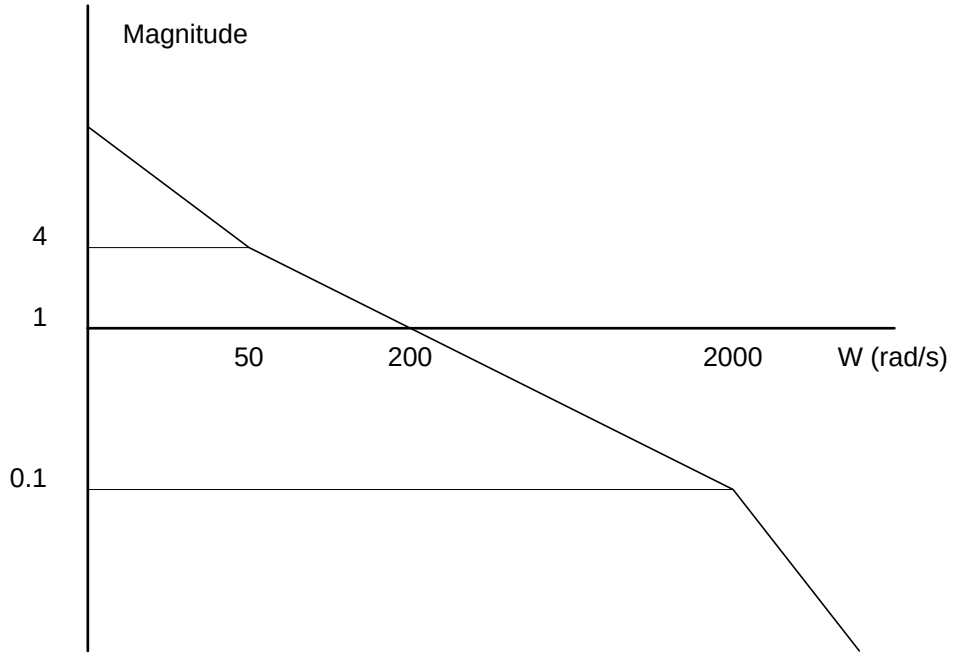


Figure 10.8 - Bode plot of the open loop transfer function

For the given example, the crossover frequency was computed numerically resulting in 200 rad/s.

Next, we determine the phase of A(s) at the crossover frequency.

$$A(j200) = 390{,}000\ (j200+51)/[(j200)^2\ .\ (j200 + 2000)]$$

$$\alpha = \text{Arg}[A(j200)] = \tan^{-1}(200/51) - 180° - \tan^{-1}(200/2000)$$

$$\alpha = 76° - 180° - 6° = -110°$$

Finally, the phase margin, PM, equals

$$PM = 180° + \alpha = 70°$$

As long as PM is positive, the system is stable. However, for a well damped system, PM should be between 30 degrees and 45 degrees. The phase margin of 70 degrees given above indicated overdamped response.

Next, we discuss the design of control systems.

System Design and Compensation

The closed-loop control system can be stabilized by a digital filter, which is preprogrammed in the DMC-2x00 controller. The filter parameters can be selected by the user for the best compensation. The following discussion presents an analytical design method.

The Analytical Method

The analytical design method is aimed at closing the loop at a crossover frequency, ω_c, with a phase margin PM. The system parameters are assumed known. The design procedure is best illustrated by a design example.

Consider a system with the following parameters:

K_t	Nm/A	Torque constant
$J = 2.10^{-4}$	$kg.m^2$	System moment of inertia
R = 2	Ω	Motor resistance
$K_a = 2$	Amp/Volt	Current amplifier gain
N = 1000	Counts/rev	Encoder line density

The DAC of the DMC-2x00 outputs +/-10V for a 14-bit command of +/-8192 counts.

The design objective is to select the filter parameters in order to close a position loop with a crossover frequency of ω_c = 500 rad/s and a phase margin of 45 degrees.

The first step is to develop a mathematical model of the system, as discussed in the previous system.

Motor

$$M(s) = P/I = K_t/Js^2 = 1000/s^2$$

Amp

$$K_a = 2 \qquad [Amp/V]$$

DAC

$$K_d = 10/32768 = .0003$$

Encoder

$$K_f = 4N/2\pi = 636$$

ZOH

$$H(s) = 2000/(s+2000)$$

Compensation Filter

$$G(s) = P + sD$$

The next step is to combine all the system elements, with the exception of G(s), into one function, L(s).

$$L(s) = M(s)\, K_a\, K_d\, K_f\, H(s) = 3.17*10^6/[s^2(s+2000)]$$

Then the open loop transfer function, A(s), is

$$A(s) = L(s)\, G(s)$$

Now, determine the magnitude and phase of L(s) at the frequency ω_c = 500.

$$L(j500) = 3.17*10^6/[(j500)^2\ (j500+2000)]$$

This function has a magnitude of

$$|L(j500)| = 0.00625$$

and a phase

$$Arg[L(j500)] = -180° - \tan^{-1}(500/2000) = -194°$$

G(s) is selected so that A(s) has a crossover frequency of 500 rad/s and a phase margin of 45 degrees. This requires that

$$|A(j500)| = 1$$

$$Arg\ [A(j500)] = -135°$$

However, since

$$A(s) = L(s)\, G(s)$$

then it follows that G(s) must have magnitude of

$$|G(j500)| = |A(j500)/L(j500)| = 160$$

and a phase

$$\arg [G(j500)] = \arg [A(j500)] - \arg [L(j500)] = -135° + 194° = 59°$$

In other words, we need to select a filter function G(s) of the form

$$G(s) = P + sD$$

so that at the frequency ω_c =500, the function would have a magnitude of 160 and a phase lead of 59 degrees.

These requirements may be expressed as:

$$|G(j500)| = |P + (j500D)| = 160$$

and

$$\arg [G(j500)] = \tan^{-1}[500D/P] = 59°$$

The solution of these equations leads to:

$$P = 160\cos 59° = 82.4$$

$$500D = 160\sin 59° = 137$$

Therefore,

$$D = 0.274$$

and

$$G = 82.4 + 0.2744s$$

The function G is equivalent to a digital filter of the form:

$$D(z) = 4KP + 4KD(1-z^{-1})$$

where

$$P = 4 * KP$$

$$D = 4 * KD * T$$

and

$$4 * KD = D/T$$

Assuming a sampling period of T=1ms, the parameters of the digital filter are:

$$KP = 20.6$$

$$KD = 68.6$$

The DMC-2x00 can be programmed with the instruction:

KP 20.6

KD 68.6

In a similar manner, other filters can be programmed. The procedure is simplified by the following table, which summarizes the relationship between the various filters.

Equivalent Filter Form

	DMC-2x00
Digital	$D(z) = [K(z-A/z) + Cz/(z-1)] * (1-B)/(Z-B)$
Digital	$D(z) = [4\ KP + 4\ KD(1-z^{-1}) + KI/2(1-z^{-1})] * (1-B)/(Z-B)$
KP, KD, KI, PL	$K = (KP + KD) \cdot 4$
	$A = KD/(KP+KD)$
	$C = KI/2$
	$B = PL$
Continuous	$G(s) = (P + Ds + I/s) * a/S+a$
PID, T	$P = 4\ KP$
	$D = 4\ T*KD$
	$I = KI/2T$
	$a = 1/T \ln (1/PL)$

Appendices

Electrical Specifications

Servo Control

ACMD Amplifier Command:	+/-10 Volts analog signal. Resolution 16-bit DAC or .0003 Volts. 3 mA maximum
A+,A-,B+,B-,IDX+,IDX- Encoder	TTL compatible, but can accept up to +/-12 Volts. Quadrature phase on CHA,CHB. Can accept single-ended (A+,B+ only) or differential (A+,A-,B+,B-). Maximum A,B edge rate: 12MHz. Minimum IDX pulse width: 80 nsec.

Input/Output

Uncommitted Inputs, Limits, Home, Abort Inputs:	TTL Can accept up to +12V signal.
OUT[1] thru OUT[3] Outputs:	TTL.

Power Requirements

+5V	400 mA
+12V	40 mA
-12V	40mA

Performance Specifications

Minimum Servo Loop Update Time:	
DMC-3415	250 μsec / 125usec with fast firmware (fast firmware only works on SLAVE controller)
DMC-3425	375 μsec / 250 usec with fast firmware (fast firmware only works on SLAVE controller)
Position Accuracy:	+/-1 quadrature count
Velocity Accuracy:	

Long Term	Phase-locked, better than .005%
Short Term	System dependent
Position Range:	+/-2147483647 counts per move
Velocity Range:	Up to 12,000,000 counts/sec servo; 3,000,000 pulses/sec-stepper
Velocity Resolution:	2 counts/sec
Motor Command Resolution:	16 bit or 0.0003 V
Variable Size:	126 user variables
Variable Range:	+/-2 billion
Variable Resolution:	$1 \cdot 10^{-4}$
Array Size:	2000 elements, 14 arrays
Program Size:	500 lines x 80 characters

Connectors for DMC-3425

J3 DMC-3425 General I/O; 37- PIN D-type

1 Reset [1]	20 Error
2 Amp Enable	21 ACMDA (PWMA)
3 Output 3	22 Output 2
4 Output 1	23 Circular Compare
5 Analog 1	24 Analog 2
6 Main Index B (Input 7) [1,2,3]	25 Home B (Input 6) [1,3]
7 Reverse Limit B (Input 5) [1,3]	26 Forward Limit B (Input 4) [1,3]
8 Input 3 [1]	27 Input 2 (and B latch) [1]
9 Input 1 (and A latch) [1]	28 Forward Limit A [1]
10 + 5V	29 Reverse Limit A [1]
11 Ground	30 Home A [1]
12 +12V	31 -12v
13 Ground	32 A Encoder A+
14 A Encoder A-	33 A Encoder B+
15 A Encoder B-	34 A Encoder Index+
16 A Encoder Index-	35 B Encoder A+
17 B Encoder A-	36 B Encoder B+
18 B Encoder B-	37 Abort [1]
19 ACMDY (SIGNA)	

J3 DMC-3425-Stepper General I/O; 37- PIN D-type

1 Reset [1]	20 PWMB
2 SIGNB	21 PWMA
3 Output 3	22 Output 2
4 Output 1	23 Circular Compare
5 Analog 1	24 Analog 2
6 Main Index B (Input 7) [1,2,3]	25 Home B (Input 6) [1,3]
7 Reverse Limit B (Input 5) [1,3]	26 Forward Limit B (Input 4) [1,3]
8 Input 3 [1]	27 Input 2 (and B latch) [1]
9 Input 1 (and A latch) [1]	28 Forward Limit A [1]
10 + 5V	29 Reverse Limit A [1]
11 Ground	30 Home A [1]
12 +12V	31 -12v
13 Ground	32 A Encoder A+
14 A Encoder A-	33 A Encoder B+
15 A Encoder B-	34 A Encoder Index+
16 A Encoder Index-	35 B Encoder A+
17 B Encoder A-	36 B Encoder B+
18 B Encoder B-	37 Abort [1]
19 SIGNA	

[1] These inputs are TTL active low and will be activated when set to 0V.
[2] All inputs are the same in terms of input range (+/-12); D13 can be used as Index B.
[3] Pins 6, 7, 25 and 26 represent Index B, Home B, Reverse Limit B and Forward Limit B. The states of these inputs are mapped to inputs 7, 6, 5 and 4 respectively. Standard input interrogation commands can be used to read these inputs (TI, MG@IN[n]), as well as the TS and MG@LFB or MG@LRB switch commands.

J5 POWER; 6 PIN MOLEX

1 +12V
2 +5V
3 +5V
4 Ground
5 Ground
6 -12V

J1 RS232 Main port: DB-9 Pin Male:

PC	Galil
1 DCD	1 RTS
2 RX	2 TX
3 TX	3 RX
4 DTR	4 CTS
5 GND	5 GND
6 DSR	6 RTS
7 RTS	7 CTS
8 CTS	8 RTS
9 RI	9 --

Pin-Out Description

OUTPUTS	DESCRIPTION
Analog Motor Command	+/- 10 Volt range signal for driving amplifier. In servo mode, motor command output is updated at the controller sample rate. In the motor off mode, this output is held at the OF command level.
Amp Enable	Signal to disable and enable an amplifier. Amp Enable goes low on Abort and OE1.
Error	The signal goes low when the position error on any axis exceeds the value specified by the error limit command, ER.
Output 1-Output 3	These 3 TTL outputs are uncommitted and may be designated by the user to toggle relays and trigger external events. The output lines are toggled by Set Bit, SB, and Clear Bit, CB, instructions. The OP instruction is used to define the state of all the bits of the Output port.

INPUTS	DESCRIPTION
Encoder, A+, B+	Position feedback from incremental encoder with two channels in quadrature, CHA and CHB. The encoder may be analog or TTL. Any resolution encoder may be used as long as the maximum frequency does not exceed 12,000,000 quadrature states/sec. The controller performs quadrature decoding of the encoder signals resulting in a resolution of quadrature counts (4 x encoder cycles). Note: Encoders that produce outputs in the format of pulses and direction may also be used by inputting the pulses into CHA and direction into Channel B and using the CE command to configure this mode.
A and B axis Encoder Index, I+	Once-Per-Revolution encoder pulse. Used in Homing sequence or Find Index command to define home on an encoder index.
A and B axis Encoder, A-, B-, I-	Differential inputs from encoder. May be input along with CHA, CHB for noise immunity of encoder signals. The CHA- and CHB- inputs are optional.

Abort input	A low input stops commanded motion instantly without a controlled deceleration. Also aborts motion program.
Reset input	A low input resets the state of the processor to its power-on condition. The previously saved state of the controller, along with parameter values, and saved sequences are restored.
Forward Limit Switch	When active, inhibits motion in forward direction. Also causes execution of limit switch subroutine, #LIMSWI. The polarity of the limit switch may be set with the CN command.
Reverse Limit Switch	When active, inhibits motion in reverse direction. Also causes execution of limit switch subroutine, #LIMSWI. The polarity of the limit switch may be set with the CN command.
Home Switch	Input for Homing (HM) and Find Edge (FE) instructions. Upon BG following HM or FE, the motor accelerates to slew speed. A transition on this input will cause the motor to decelerate to a stop. The polarity of the Home Switch may be set with the CN command.
Input 1 - Input 3	Uncommitted inputs. May be defined by the user to trigger events. Inputs are checked with the Conditional Jump instruction and After Input instruction or Input Interrupt. Input 1 is used for the high-speed latch. Only 3 inputs for the DMC-3425.
Latch input	High speed position latch to capture axis position within 20 nano seconds on occurrence of latch signal. AL command arms latch. Input 1 is latch for A axis. Input 2 is latch for B axis if using DMC-3425
Analog input	12 bit resolution

ICM-1460 Interconnect Module

The ICM-1460, Rev F Interconnect Module provides easy connections between the DMC-3425 series controllers and other system elements, such as amplifiers, encoders, and external switches. The ICM-1460 accepts the 37-pin cable from the DMC-3425 and provides screw-type terminals. Each screw terminal is labeled for quick connection of system elements.

The ICM-1460 is packaged as a circuit board mounted to a metal enclosure. A version of the ICM-1460 is also available with a single PWM brush servo amplifier, or with a 20W linear brush servo amplifier. (see AMP-1460 and ICM-1460-20W).

Features

- Breaks out 37-pin ribbon cable into individual screw-type terminals.
- Clearly identifies all terminals
- Available with on-board servo drive (see AMP-1460 or ICM-1460-20W).
- 10-pin IDC connectors for encoders.
- Option for Opto-isolation of all general purpose inputs, committed inputs, and digital outputs. Specify at time of order with the –OPTO option.

Specifications

Dimensions: 6.9" x 4.9" x 2.6" Weight: 1 pound

Rev A-F Terminal#	Rev G Terminal#	Label	I/O	Description
1	1	+12V[4]	O	+12 Volts
2	2	-12V[4]	O	-12 Volts
3	3	AMPEN/SIGNY[5]	O	Amplifier enable X axis or Y Axis Sign Output for Stepper
4	4	ACMDX/PULSE(X)	O	X Axis Motor command or Pulse Output for Stepper
5	5	AN1	O	Analog Input 1
6	6	AI2	O	Analog Input 2
7	7	GND	--	Signal Ground
8	8	RESET	I	Reset
9	9	ERROR/PULSE(Y) [6]	O	Error signal or Y Axis Pulse Output for Stepper
10	10	OUT3	O	Output 3
11	11	OUT2	O	Output 2
12	12	OUT1	O	Output 1
13	13	CMP/ICOM [7]	O	Circular Compare / Input common for Opto option
14	14	5V	O	+ 5 Volts
15	15	GND	--	Signal Ground
16	16	IN7/INDY+	I	Input 7 (Y Axis Main Encoder Index + for DMC-1425)
17	17	IN6/HOMY	I	Input 6 (Y Axis Home input for DMC-1425)
18	18	IN5/RLSY	I	Input 5 (Y axis reverse limit on DMC-1425)
19	19	IN4/FLSY	I	Input 4 (Y axis forward limit on DMC-1425)
20	20	IN3/IDY-	I	Input 3 (Y axis main encoder index for DMC-1425)
21	21	IN2	I	Input 2
22	22	IN1/LTCH	I	Input 1 / Input for Latch Function
23	23	FLSX	I	Forward limit switch input
24	24	RLSX	I	Reverse limit switch input
25	25	HOMX	I	Home input
26	26	ABORT	I	Abort Input
27	27	GND	--	Signal Ground
28	28	MA+	I	X Axis Main Encoder A+ [5]
29	29	MA-	I	X Axis Main Encoder A- [5]
30	30	MB+	I	X Axis Main Encoder B+ [5]
31	31	MB-	I	X Axis Main Encoder B- [5]
32	32	IDX+	I	X Axis Main Encoder Index + [5]
33	33	IDX-	I	X Axis Main Encoder Index – [5]
34	34	AA+	I	X Axis Auxiliary Encoder A+ (Y Axis Main Encoder A+ for DMC-1425)
35	35	AA-	I	X Axis Auxiliary Encoder A- (Y Axis Main Encoder A- for DMC-1425)
36	36	AB+	I	X Axis Auxiliary Encoder B+ (Y Axis Main Encoder B+ for DMC-1425)
37	37	AB-	I	X Axis Auxiliary Encoder B- (Y Axis Main Encoder B- for DMC-1425)
8	38	ACMD2/SIGNX	O	2nd Motor command Signal for Sine Amplifier or SIGNX for stepper
39	39	5V	O	+ 5 Volts
40	40	GND	--	Signal Ground

[4]The screw terminals for ACMDX and ACMDY can provide access to 2 sets of signals, depending on the placement of the 2 jumpers on JP3.
[5]If the Opto-isolated input option is used, the output compare is NOT brought out to the ICM-1460. If the output compare is to be used in conjunction with the opto-isolation, pin 23 of the Cable 37-Pin D must be brought out externally. There are also options for using either terminal 1 or 2 as the Common connection. Contact Galil for more information.

Opto-Isolation Option for ICM-1460

The ICM-1460 module from Galil has an option for opto-isolated inputs and outputs. This option is specified as ICM-1460-OPTO*. With this option, the user is able to use voltages up to 24V on the inputs and outputs of the controller.

The common point for the opto-isolation may be chosen from any of the following pins: pin 1 (labeled as +12V), pin 2 (labeled as –12V) or pin 13 (labeled as CMP/ICOM). When pin 1 is used as input/output common, the +12V output be comes inaccessible, when pin 2 is used, the –12V becomes inaccessible, and when pin 13 is used, the output compare function is not available. This common point must be specified at the time of ordering.

The ICM-1460 may also be configured such that the input/output common is jumpered to the internal Vcc (+5V). By doing this, no screw connection is needed so no signals are lost.

A final option for the opto-isolation is for separate input/output commons. This allows the user to have different voltage levels for the inputs and outputs. However, this requires the use of both pin 1 and pin 2 on the screw connection, making both +12V and –12V inaccessible on the screw terminals.

Opto-isolated inputs:

The signal "IN[x]" below is one of the isolated digital inputs where x stands for the digital input terminals.

By connecting the OPTO-COMMON to the + side of an isolated power supply, the inputs will be activated by sinking current. By connecting the OPTO-COMMON to the GND side of the power supply, the inputs will be activated by sourcing current.

The opto-isolation circuit requires 1ma drive current with approximately 400 usec response time. The voltage should not exceed 24V without placing additional resistance to limit the current to 11 mA.

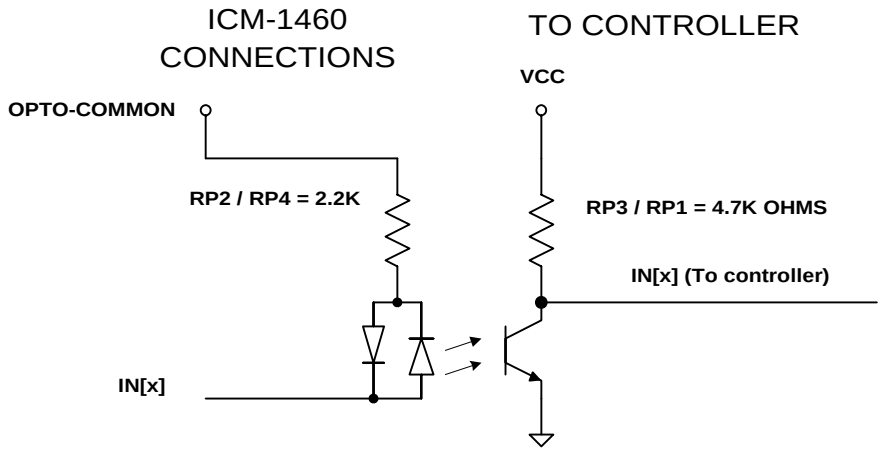


Figure A-1

Opto-isolated outputs:

The signal “OUT[x]" below is one of the isolated digital outputs where x stands for the digital output terminals.

The OPTO-COMMON needs to be connected to an isolated power supply. The OUT[x] can be used to source current from the power supply. The maximum sourcing current for the OUT[x] is 25 ma. Sinking configuration can also be specified. Please contact Galil for details.

The default state of the outputs may also be set through the resistor pack RP5. With this resistor in it’s default state, the opto-isolator will be ON. By reversing RP5 in its socket, the opto-isolator will be OFF by default.

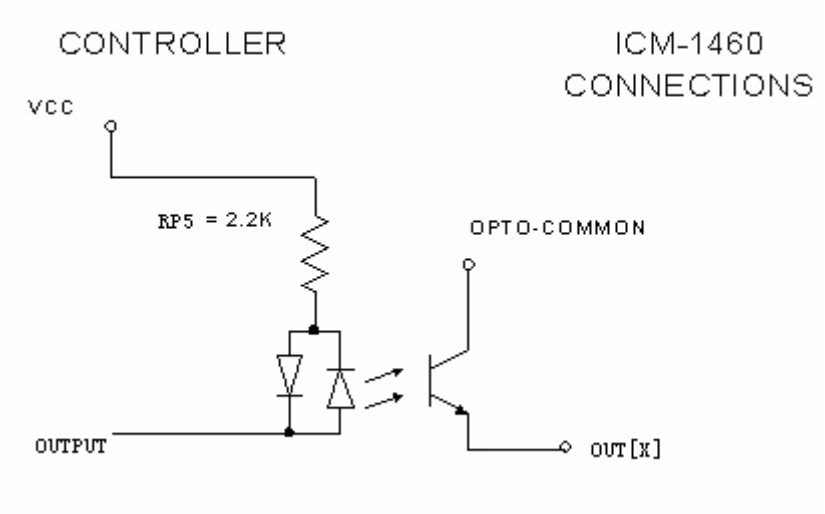


Figure A-2

* Only available with ICM-1460

64 Extended I/O of the DMC-3425 Controller

The DMC-3425 controller offers 64 extended I/O points, which can be interfaced to Grayhill and OPTO-22 I/O mounting racks. These I/O points can be configured as inputs or outputs in 8 bit increments through software. The I/O points are accessed through two 50-pin IDC connectors, each with 32 I/O points.

Configuring the I/O of the DMC-3425 with DB-14064

The 64 extended I/O points of the DMC-3425 w/DB-14064 series controller can be configured in blocks of 8. The extended I/O is denoted as blocks 2-9 or bits 17-80.

The command, CO, is used to configure the extended I/O as inputs or outputs. The CO command has one field:

CO n

Where, n is a decimal value, which represents a binary number. Each bit of the binary number represents one block of extended I/O. When set to 1, the corresponding block is configured as an output.

Note: The CO command must be sent through the SA command to configure outputs for slave controller extended I/O.

The least significant bit represents block 2 and the most significant bit represents block 9. The decimal value can be calculated by the following formula. $n = n_2 + 2*n_3 + 4*n_4 + 8*n_5 +16* n_6 +32* n_7 +64* n_8 +128* n_9$ where n_x represents the block. If the n_x value is a one, then the block of 8 I/O points is to be configured as an output. If the n_x value is a zero, then the block of 8 I/O points will be configured as an input. For example, if block 4 and 5 is to be configured as an output, CO 12 is issued.

8-Bit I/O Block	Block	Binary Representation	Decimal Value for Block
17-24	2	2^0	1
25-32	3	2^1	2
33-40	4	2^2	4
41-48	5	2^3	8
49-56	6	2^4	16
57-64	7	2^5	32
65-72	8	2^6	64
73-80	9	2^7	128

The simplest method for determining n:

Step 1. Determine which 8-bit I/O blocks to be configured as outputs.

Step 2. From the table, determine the decimal value for each I/O block to be set as an output.

Step 3. Add up all of the values determined in step 2. This is the value to be used for n.

For example, if blocks 2 and 3 are to be outputs, then n is 3 and the command, CO3, should be issued. Note: This calculation is identical to the formula: $n = n_2 + 2*n_3 + 4*n_4 + 8*n_5 +16* n_6 +32* n_7 +64* n_8 +128* n_9$ where n_x represents the block.

Saving the State of the Outputs in Non-Volatile Memory

The configuration of the extended I/O and the state of the outputs can be stored in the EEPROM with the BN command. If no value has been set, the default of CO 0 is used (all blocks are inputs).

Accessing extended I/O

When configured as an output, each I/O point may be defined with the SBn and CBn commands (where n=1 through 8 and 17 through 80). Outputs may also be defined with the conditional command, OBn (where n=1 through 8 and 17 through 80).

The command, OP, may also be used to set output bits, specified as blocks of data. The OP command accepts 5 parameters. The first parameter sets the values of the main output port of the controller (Outputs 1-8, block 0). The additional parameters set the value of the extended I/O as outlined:

OP m,a,b,c,d

where m is the decimal representation of the bits 1-8 (values from 0 to 255) and a,b,c,d represent the extended I/O in consecutive groups of 16 bits. (values from 0 to 65535). Arguments that are given for I/O points configured as inputs will be ignored. The following table describes the arguments used to set the state of outputs.

Argument	Blocks	Bits	Description
M	0	1-8	General Outputs
A	2,3	17-32	Extended I/O
B	4,5	33-48	Extended I/O
C	6,7	49-64	Extended I/O
D	8,9	65-80	Extended I/O

For example, if block 8 is configured as an output, the following command may be issued:

```
OP 7,,,,7
```

This command will set bits 1,2,3 (block 0) and bits 65,66,67 (block 8) to 1. Bits 4 through 8 and bits 68 through 80 will be set to 0. All other bits are unaffected.

When accessing I/O blocks configured as inputs, use the TIn command. The argument 'n' refers to the block to be read (n=0,2,3,4,5,6,7,8 or 9). The value returned will be a decimal representation of the corresponding bits.

Individual bits can be queried using the @IN[n] function (where n=1 through 8 or 17 through 80). If the following command is issued;

```
MG @IN[17]
```

the controller will return the state of the least significant bit of block 2 (assuming block 2 is configured as an input).

Connector Description:

The DMC-3425 controller with DB-14064 has two 50 Pin IDC header connectors. The connectors are compatible with I/O mounting racks such as Grayhill 70GRCM32-HL and OPTO-22 G4PB24.

Note for interfacing to OPTO-22 G4PB24: When using the OPTO-22 G4PB24 I/O mounting rack, the user will only have access to 48 of the 64 I/O points available on the controller. Block 5 and Block 9 must be configured as inputs and will be grounded by the I/O rack.

J6 50-PIN IDC

Pin	Signal	Block	Bit @IN[n], @OUT[n]	Bit No
1.	I/O	4	40	7
3.	I/O	4	39	6
5	I/O	4	38	5
7.	I/O	4	37	4
9.	I/O	4	36	3
11.	I/O	4	35	2
13.	I/O	4	34	1
15.	I/O	4	33	0
17.	I/O	3	32	7
19.	I/O	3	31	6
21.	I/O	3	30	5
23.	I/O	3	29	4
25.	I/O	3	28	3
27.	I/O	3	27	2
29.	I/O	3	26	1
31.	I/O	3	25	0
33.	I/O	2	24	7
35.	I/O	2	23	6
37.	I/O	2	22	5
39.	I/O	2	21	4
41.	I/O	2	20	3
43.	I/O	2	19	2
45.	I/O	2	18	1
47.	I/O	2	17	0
49.	+5V	-	-	-

2.	I/O	5	48	0
4.	I/O	5	47	1
6.	I/O	5	46	2
8.	I/O	5	45	3
10.	I/O	5	44	4
12.	I/O	5	43	5
14.	I/O	5	42	6
16.	I/O	5	41	7
18.	GND	-	-	-
20.	GND	-	-	-
22.	GND	-	-	-
24.	GND	-	-	-
26.	GND	-	-	-
28.	GND	-	-	-
30.	GND	-	-	-
32.	GND	-	-	-
34.	GND	-	-	-
36.	GND	-	-	-
38.	GND	-	-	-
40.	GND	-	-	-
42.	GND	-	-	-
44.	GND	-	-	-
46.	GND	-	-	-
48.	GND	-	-	-
50.	GND	-	-	-

J8 50-PIN IDC

Pin	Signal	Block	Bit @IN[n], @OUT[n]	Bit No
1.	I/O	8	72	7
3.	I/O	8	71	6
5	I/O	8	70	5
7.	I/O	8	69	4
9.	I/O	8	68	3
11.	I/O	8	67	2
13.	I/O	8	66	1
15.	I/O	8	65	0
17.	I/O	7	64	7
19.	I/O	7	63	6
21.	I/O	7	62	5
23.	I/O	7	61	4
25.	I/O	7	60	3
27.	I/O	7	59	2
29.	I/O	7	58	1
31.	I/O	7	57	0
33.	I/O	6	56	7
35.	I/O	6	55	6
37.	I/O	6	54	5
39.	I/O	6	53	4
41.	I/O	6	52	3
43.	I/O	6	51	2
45.	I/O	6	50	1
47.	I/O	6	49	0
49.	+5V	-	-	-

2.	I/O	9	80	7
4.	I/O	9	79	6
6.	I/O	9	78	5
8.	I/O	9	77	4
10.	I/O	9	76	3
12.	I/O	9	75	2
14.	I/O	9	74	1
16.	I/O	9	73	0
18.	GND	-	-	-
20.	GND	-	-	-
22.	GND	-	-	-
24.	GND	-	-	-
26.	GND	-	-	-
28.	GND	-	-	-
30.	GND	-	-	-
32.	GND	-	-	-
34.	GND	-	-	-
36.	GND	-	-	-
38.	GND	-	-	-
40.	GND	-	-	-
42.	GND	-	-	-
44.	GND	-	-	-
46.	GND	-	-	-
48.	GND	-	-	-
50.	GND	-	-	-

IOM-1964 Opto-Isolation Module for Extended I/O Controllers

Description:

- Provides 64 optically isolated inputs and outputs, each rated for 2mA at up to 28 VDC
- Configurable as inputs or outputs in groups of eight bits
- Provides 16 high power outputs capable of up to 500mA each
- Connects to controller via 100 pin shielded cable
- All I/O points conveniently labeled
- Each of the 64 I/O points has status LED
- Dimensions 6.8” x 11.4”
- Works with extended I/O controllers

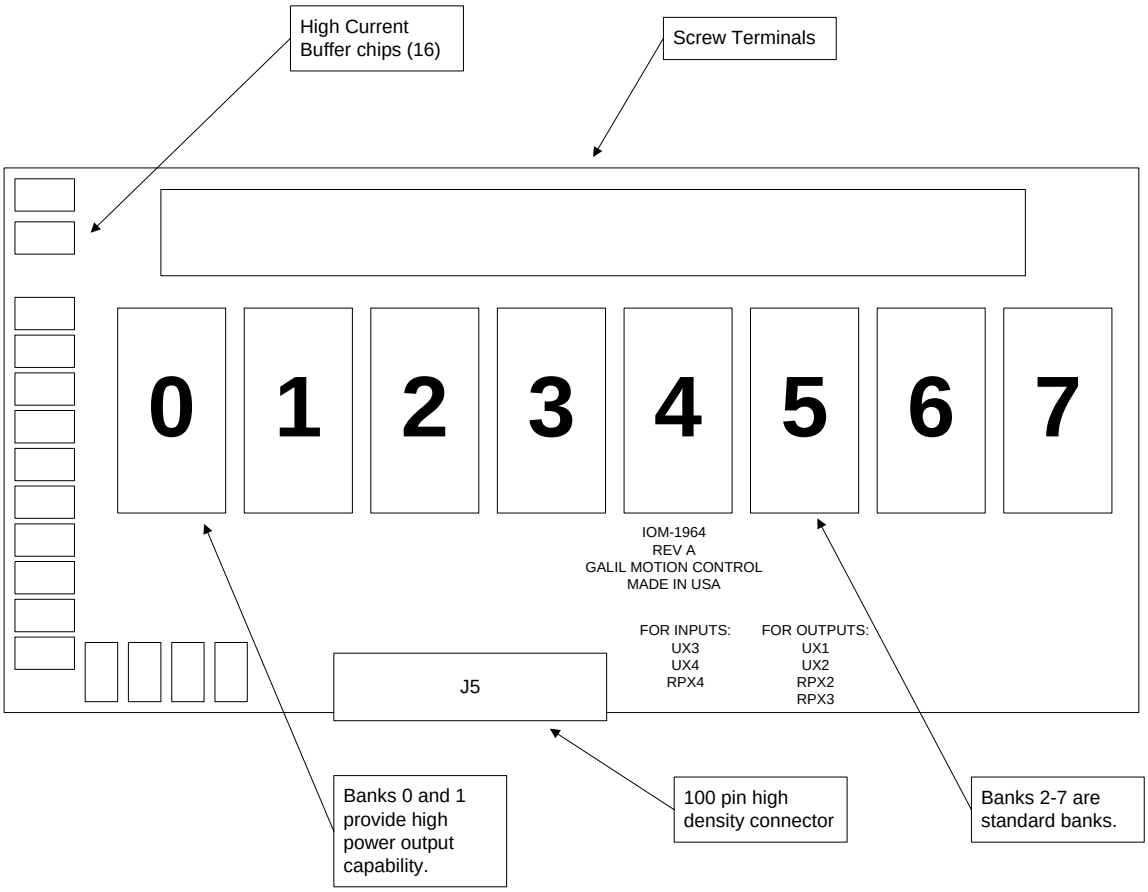


Figure A-3

Overview

The IOM-1964 is an input/output module that connects to the DB-14064 extended I/O daughter board cards from Galil, providing optically isolated buffers for the extended inputs and outputs of the controller. The IOM-1964 also provides 16 high power outputs capable of 500mA of current per output point. The IOM-1964 splits the 64 I/O points into eight banks of eight I/O points each, corresponding to the eight banks of extended I/O on the controller. Each bank is individually configured as an input or output bank by inserting the appropriate integrated circuits and resistor packs. The hardware configuration of the IOM-1964 must match the software configuration of the controller card.

All E-Series controllers have general purpose I/O connections. On a DMC-3425 and 3415 the standard uncommitted I/O consists of: three TTL digital inputs, three TTL digital outputs, and two analog inputs.

The DMC-34x5 with the DB-14064, however, has an additional 64 digital input/output points than the 6 described above for a total of 70 input/output points. The 64 I/O points on the DB-14064 are attached via two 50-pin ribbon cable header connectors. A CB-50-80 adapter card is used to connect the two 50-pin ribbon cables to an 80-pin high-density connector identical to the main axes connector. An 80-pin shielded cable connects from the 80-pin connector of the CB-50-80 board to the 80-pin high density connector J5 on the IOM-1964.

Configuring Hardware Banks

The extended I/O on the DMC-34x5 with DB-14064 is configured using the CO command. The banks of buffers on the IOM-1964 are configured to match by inserting the appropriate IC's and resistor packs. The layout of each of the I/O banks is identical.

For example, here is the layout of bank 0:

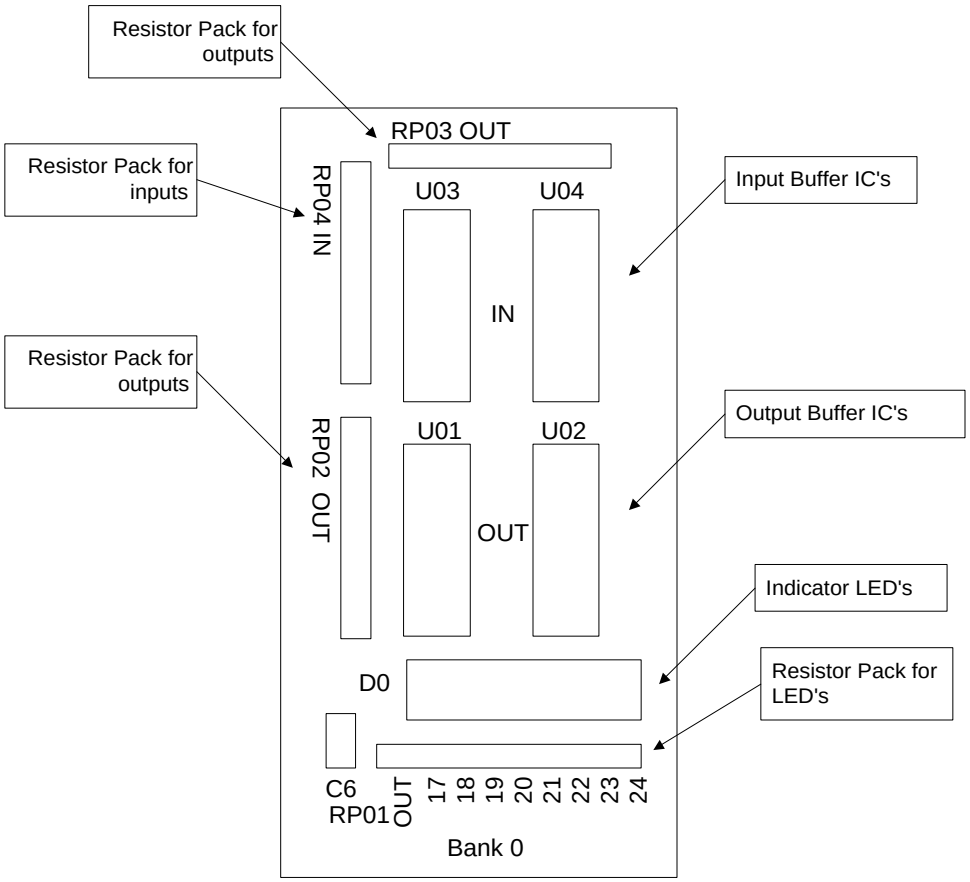


Figure A-4

All of the banks have the same configuration pattern as diagrammed above in figure A-4. For example, all banks have Ux1 and Ux2 output optical isolator IC sockets, labeled in bank 0 as U01 and U02, in bank 1 as U11 and U12, and so on. Each bank is configured as inputs or outputs by inserting optical isolator IC's and resistor packs in the appropriate sockets. A group of eight LED's indicates the status of each I/O point. The numbers above the Bank 0 label indicate the number of the I/O point corresponding to the LED above it.

Digital Inputs

Configuring a bank for inputs requires that the Ux3 and Ux4 sockets be populated with NEC2505 optical isolation integrated circuits. The IOM-1964 is shipped with a default configuration of banks 2-7 configured as inputs. The output IC sockets Ux1 and Ux2 must be empty. The input IC's are labeled Ux3 and Ux4. For example, in bank 0 the IC's are U03 and U04, bank 1 input IC's are labeled U13 and U14, and so on. Also, the resistor pack RPx4 must be inserted into the bank to finish the input configuration.

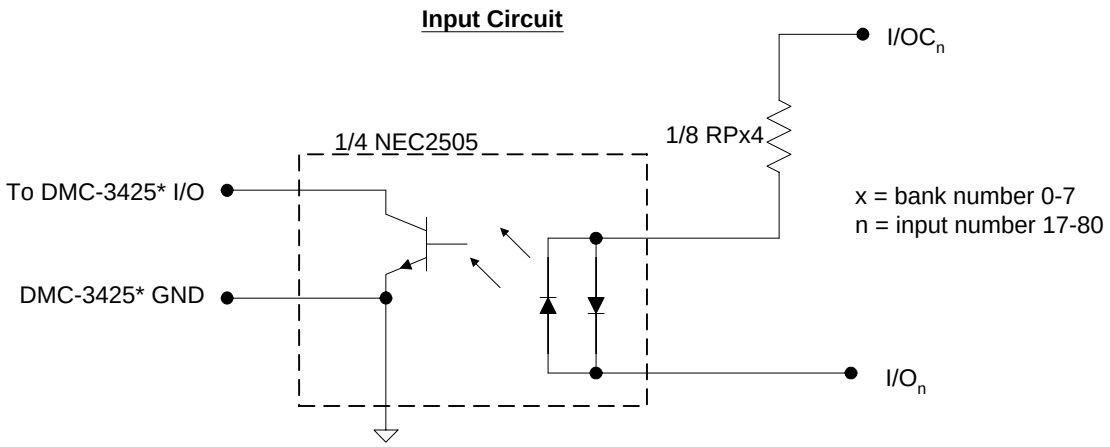
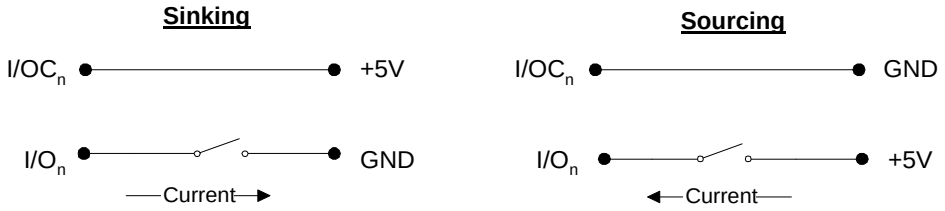


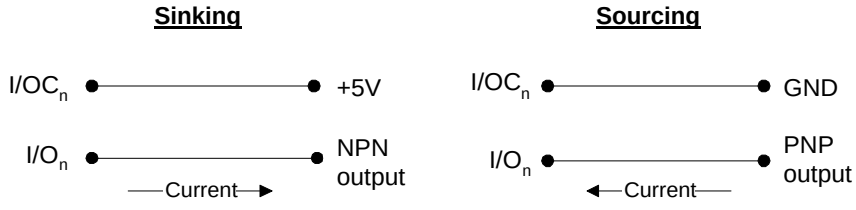
Figure A-5 – Input Circuit

Connections to this optically isolated input circuit are done in a sinking or sourcing configuration, referring to the direction of current. Some example circuits are shown below:



There is one I/OC connection for each bank of eight inputs. Whether the input is connected as sinking or sourcing, when the switch is open no current flows and the digital input function @IN[n] returns 1. This is because of an internal pull up resistor on the DB-14064. When the switch is closed in either circuit, current flows. This pulls the input on the DB-14064 to ground, and the digital input function @IN[n] returns 0. Note that the external +5V in the circuits above is for example only. The inputs are optically isolated and can accept a range of input voltages from 4 to 28 VDC.

Active outputs are connected to the optically isolated inputs in a similar fashion with respect to current. An NPN output is connected in a sinking configuration, and a PNP output is connected in the sourcing configuration.



Whether connected in a sinking or sourcing circuit, only two connections are needed in each case. When the NPN output is 5 volts, then no current flows and the input reads 1. When the NPN output goes to 0 volts, then it sinks current and the input reads 0. The PNP output works in a similar fashion, but the voltages are reversed i.e. 5 volts on the PNP output sources current into the digital input and the input reads 0. As before, the 5 volt is an example, the I/OC can accept between 4-28 volts DC.

Note that the current through the digital input should be kept below 3 mA in order to minimize the power dissipated in the resistor pack. This will help prevent circuit failures. The resistor pack RPx4 is standard 1.5k ohm that is suitable for power supply voltages up to 5.5 VDC. However, use of 24 VDC for example would require a higher resistance such as a 10k ohm resistor pack.

High Power Digital Outputs

The first two banks on the IOM-1964, banks 0 and 1, have high current output drive capability. The IOM-1964 is shipped with banks 0 and 1 configured as outputs. Each output can drive up to 500mA of continuous current. Configuring a bank of I/O as outputs is done by inserting the optical isolator NEC2505 IC's into the Ux1 and Ux2 sockets. The digital input IC's Ux3 and Ux4 are removed. The resistor packs RPx2 and RPx3 are inserted, and the input resistor pack RPx4 is removed.

Each bank of eight outputs shares one I/OC connection, which is connected to a DC power supply between 4 and 28 VDC. A 10k ohm resistor pack should be used for RPx3. Here is a circuit diagram:

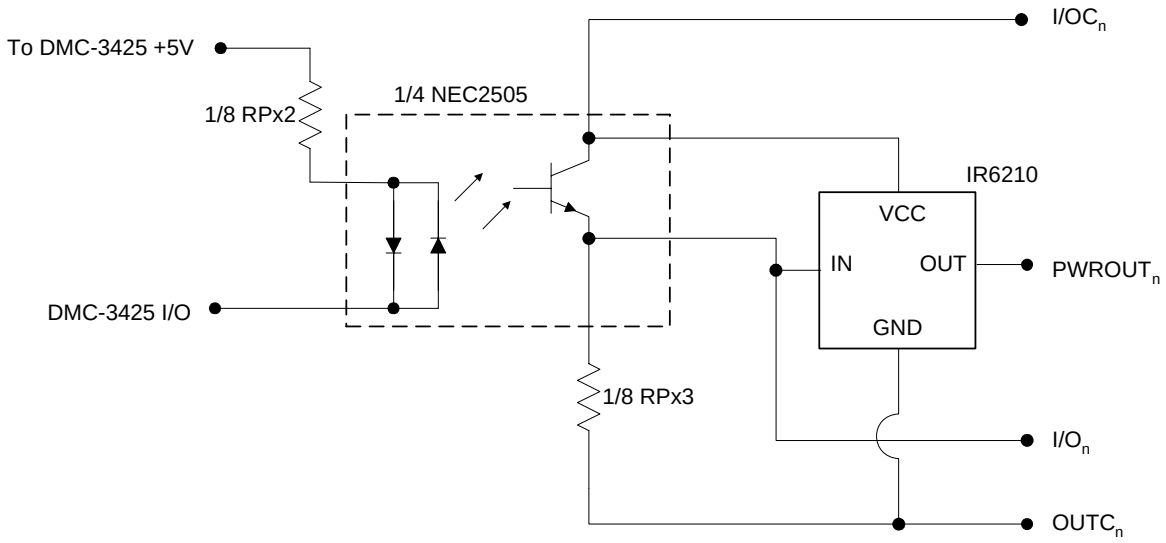


Figure A-6

The load is connected between the power output and output common. The I/O connection is for test purposes, and would not normally be connected. An external power supply is connected to the I/OC and OUTC terminals, which isolates the circuitry of the DMC-3425 controller/DB-14064 daughter board from the output circuit.

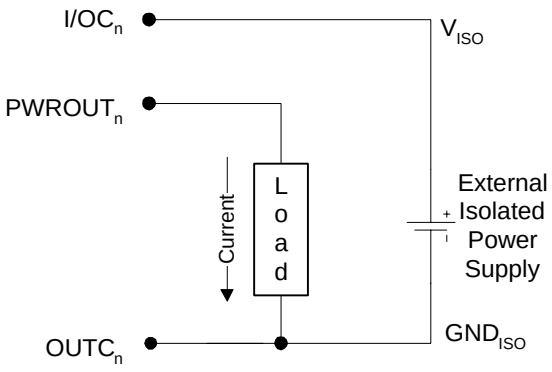


Figure A-7

The power outputs must be connected in a driving configuration as shown on the previous page. Here are the voltage outputs to expect after the Clear Bit and Set Bit commands are given:

Output Command	Result
CB_n	$V_{pwr} = V_{iso}$
SB_n	$V_{pwr} = GND_{iso}$

Standard Digital Outputs

The I/O banks 2-7 can be configured as optically isolated digital outputs, however these banks do not have the high power capacity as in banks 0-1. In order to configure a bank as outputs, the optical isolator chips Ux1 and Ux2 are inserted, and the digital input isolator chips Ux3 and Ux4 are removed. The resistor packs RPx2 and RPx3 are inserted, and the input resistor pack RPx4 is removed.

Each bank of eight outputs shares one I/OC connection, which is connected to a DC power supply between 4 and 28 VDC. The resistor pack RPx3 is optional, used either as a pull up resistor from the output transistor's collector to the external supply connected to I/OC or the RPx3 is removed resulting in an open collector output. Figure A-8 is a schematic of the digital output circuit:

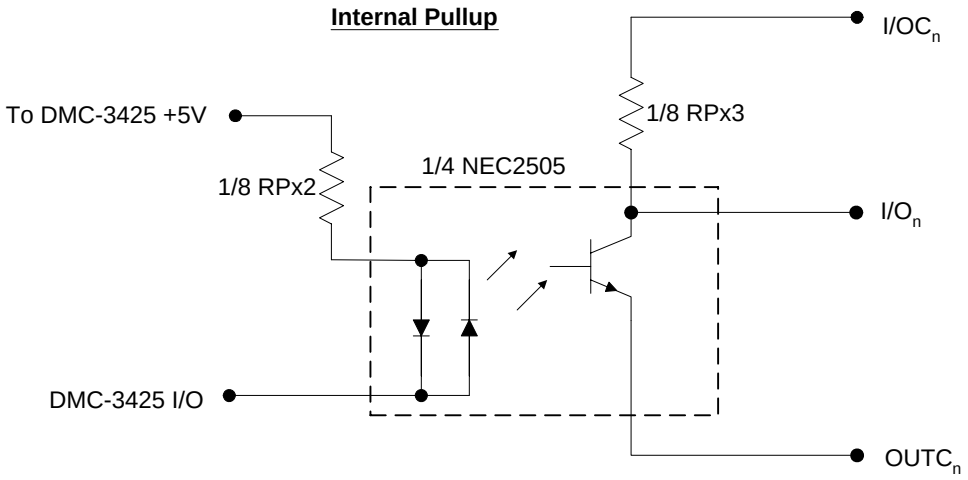


Figure A-8 – Internal Pullup

The resistor pack RPx3 limits the amount of current available to source, as well as affecting the low level voltage at the I/O output. The maximum sink current is 2mA regardless of RPx3 or I/OC voltage, determined by the NEC2505 optical isolator IC. The maximum source current is determined by dividing the external power supply voltage by the resistor value of RPx3.

The high level voltage at the I/O output is equal to the external supply voltage at I/OC. However, when the output transistor is on and conducting current, the low level output voltage is determined by three factors. The external supply voltage, the resistor pack RPx3 value, and the current sinking limit of the NEC2505 all determine the low level voltage. The sink current available from the NEC2505 is between 0 and 2mA. Therefore, the maximum voltage drop across RPx3 is calculated by multiplying the 2mA maximum current times the resistor value of RPx3. For example, if a 10k ohm resistor pack is used for RPx3, then the maximum voltage drop is 20 volts. The digital output will never drop below the voltage at OUTC, however. Therefore a 10k ohm resistor pack will result in a low level voltage of .7 to 1.0 volts at the I/O output for an external supply voltage between 4 and 21 VDC. If a supply voltage greater than 21 VDC is used, a higher value resistor pack will be required.

Output Command	Result
CB_n	$V_{out} = GND_{iso}$
SB_n	$V_{out} = V_{iso}$

The resistor pack RPx3 is removed to provide open collector outputs. The same calculations for maximum source current and low level voltage applies as in the above circuit. The maximum sink current is determined by the NEC2505, and is approximately 2mA.

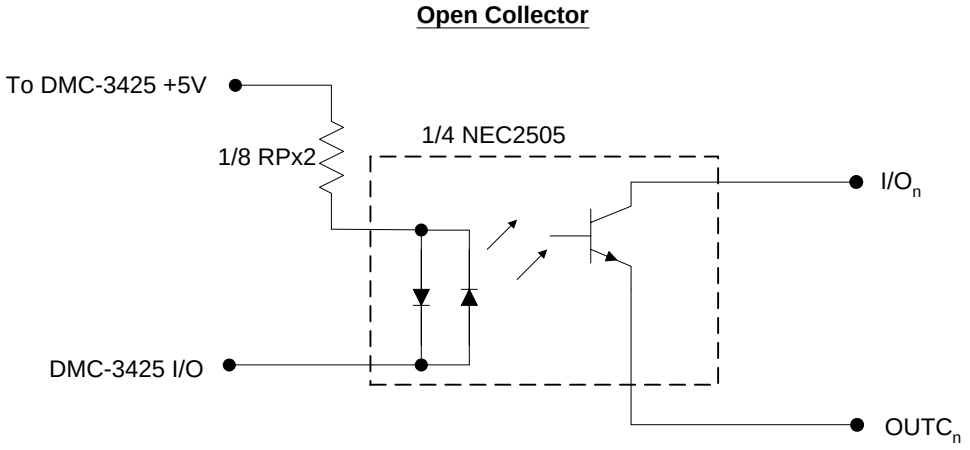


Figure A-9 – Open Collector

Electrical Specifications

- I/O points, configurable as inputs or outputs in groups of 8

Digital Inputs

- Maximum voltage: 28 VDC
- Minimum input voltage: 4 VDC
- Maximum input current: 3 mA

High Power Digital Outputs

- Maximum external power supply voltage: 28 VDC
- Minimum external power supply voltage: 4 VDC
- Maximum source current, per output: 500mA
- Maximum sink current: sinking circuit inoperative

Standard Digital Outputs

- Maximum external power supply voltage: 28 VDC
- Minimum external power supply voltage: 4 VDC
- Maximum source current: limited by pull up resistor value

- Maximum sink current: 2mA

Relevant DMC Commands

CO n	Configures the 64 bits of extended I/O in 8 banks of 8 bits each. $n = n_2 + 2*n_3 + 4*n_4 + 8*n_5 + 16*n_6 + 32*n_7 + 64*n_8 + 128*n_9$ where n_x is a 1 or 0, 1 for outputs and 0 for inputs. The x is the bank number
OP	m = 8 standard digital outputs
m,n,o,p,q	n = extended I/O banks 0 & 1, outputs 17-32 o = extended I/O banks 2 & 3, outputs 33-48 p = extended I/O banks 4 & 5, outputs 49-64 q = extended I/O banks 6 & 7, outputs 65-80
SB n	Sets the output bit to a logic 1, n is the number of the output from 1 to 80.
CB n	Clears the output bit to a logic 0, n is the number of the output from 1 to 80.
OB n,m	Sets the state of an output as 0 or 1, also able to use logical conditions.
TI n	Returns the state of 8 digital inputs as binary converted to decimal, n is the bank number +2.
_TI n	Operand (internal variable) that holds the same value as that returned by TI n.
@IN[n]	Function that returns state of individual input bit, n is number of the input from 1 to 80.

Screw Terminal Listing

Rev A+B boards (orange) and Rev C boards (black) have the pinouts listed below.

REV A+B TERMINAL #	REV C TERMINAL #	LABEL	DESCRIPTION	BANK
1		GND	Ground	N/A
2	2	5V	5V DC out	N/A
3	1	GND	Ground	N/A
4	4	5V	5V DC out	N/A
5	3	I/O80	I/O bit 80	7
6	6	I/O79	I/O bit 79	7
7	5	I/O78	I/O bit 78	7
8	8	I/O77	I/O bit 77	7
9	7	I/O76	I/O bit 76	7
10	10	I/O75	I/O bit 75	7
11	9	I/O74	I/O bit 74	7
12	12	I/O73	I/O bit 73	7
13	11	OUTC73-80	Out common for I/O 73-80	7
14	14	I/OC73-80	I/O common for I/O 73-80	7
15	13	I/O72	I/O bit 72	6
16	16	I/O71	I/O bit 71	6
17	15	I/O70	I/O bit 70	6
18	18	I/O69	I/O bit 69	6
19	17	I/O68	I/O bit 68	6
20	20	I/O67	I/O bit 67	6
21	19	I/O66	I/O bit 66	6

22	22	I/O65	I/O bit 65	6
23	21	OUTC65-72	Out common for I/O 65-72	6
24	24	I/OC65-72	I/O common for I/O 65-72	6
25	23	I/O64	I/O bit 64	5
26	26	I/O63	I/O bit 63	5
27	25	I/O62	I/O bit 62	5
28	28	I/O61	I/O bit 61	5
29	27	I/O60	I/O bit 60	5
30	30	I/O59	I/O bit 59	5
31	29	I/O58	I/O bit 58	5
32	32	I/O57	I/O bit 57	5
33	31	OUTC57-64	Out common for I/O 57-64	5
34	34	I/OC57-64	I/O common for I/O 57-64	5
35	33	I/O56	I/O bit 56	4
36	36	I/O55	I/O bit 55	4
37	35	I/O54	I/O bit 54	4
38	38	I/O53	I/O bit 53	4
39	37	I/O52	I/O bit 52	4
40	40	I/O51	I/O bit 51	4
41	39	I/O50	I/O bit 50	4
42	42	I/O49	I/O bit 49	4
43	41	OUTC49-56	Out common for I/O 49-56	4
44	44	I/OC49-56	I/O common for I/O 49-56	4
45	43	I/O48	I/O bit 48	3
46	46	I/O47	I/O bit 47	3
47	45	I/O46	I/O bit 46	3
48	48	I/O45	I/O bit 45	3
49	47	I/O44	I/O bit 44	3
50	50	I/O43	I/O bit 43	3
51	49	I/O42	I/O bit 42	3
52	52	I/O41	I/O bit 41	3
53	51	OUTC41-48	Out common for I/O 41-48	3
54	54	I/OC41-48	I/O common for I/O 41-48	3
55	53	I/O40	I/O bit 40	2
56	56	I/O39	I/O bit 39	2
57	55	I/O38	I/O bit 38	2
58	58	I/O37	I/O bit 37	2
59	57	I/O36	I/O bit 36	2
60	60	I/O35	I/O bit 35	2
61	59	I/O34	I/O bit 34	2
62	62	I/O33	I/O bit 33	2
63	61	OUTC33-40	Out common for I/O 33-40	2
64	64	I/OC33-40	I/O common for I/O 33-40	2
65	63	I/O32	I/O bit 32	1
66	66	I/O31	I/O bit 31	1

67	65	I/O30	I/O bit 30	1
68	68	I/O29	I/O bit 29	1
69	67	I/O28	I/O bit 28	1
70	70	I/O27	I/O bit 27	1
71	69	I/O26	I/O bit 26	1
72	72	I/O25	I/O bit 25	1
73	71	OUTC25-32	Out common for I/O 25-32	1
74	74	I/OC25-32	I/O common for I/O 25-32	1
75	73	OUTC25-32	Out common for I/O 25-32	1
76	76	I/OC25-32	I/O common for I/O 25-32	1
77	75	PWROUT32	Power output 32	1
78	78	PWROUT31	Power output 31	1
79	77	PWROUT30	Power output 30	1
80	80	PWROUT29	Power output 29	1
81	79	PWROUT28	Power output 28	1
82	82	PWROUT27	Power output 27	1
83	81	PWROUT26	Power output 26	1
84	84	PWROUT25	Power output 25	1
85	83	I/O24	I/O bit 24	0
86	86	I/O23	I/O bit 23	0
87	85	I/O22	I/O bit 22	0
88	88	I/O21	I/O bit 21	0
89	87	I/O20	I/O bit 20	0
90	90	I/O19	I/O bit 19	0
91	89	I/O18	I/O bit 18	0
92	92	I/O17	I/O bit 17	0
93	91	OUTC17-24	Out common for I/O 17-24	0
94	94	I/OC17-24	I/O common for I/O 17-24	0
95	93	OUTC17-24	Out common for I/O 17-24	0
96	96	I/OC17-24	I/O common for I/O 17-24	0
97	95	PWROUT24	Power output 24	0
98	98	PWROUT23	Power output 23	0
99	97	PWROUT22	Power output 22	0
100	100	PWROUT21	Power output 21	0
101	99	PWROUT20	Power output 20	0
102	102	PWROUT19	Power output 19	0
103	101	PWROUT18	Power output 18	0
104	104	PWROUT17	Power output 17	0
	103	GND	Ground	

* Silkscreen on Rev A board is incorrect for these terminals.

Coordinated Motion - Mathematical Analysis

The terms of coordinated motion are best explained in terms of the vector motion. The vector velocity, V_s, which is also known as the feed rate, is the vector sum of the velocities along the A and B axes, V_a and V_b.

$$V_s = \sqrt{V_a{}^2 + V_b{}^2}$$

The vector distance is the integral of Vs, or the total distance traveled along the path. To illustrate this further, suppose that a string was placed along the path in the A-B plane. The length of that string represents the distance traveled by the vector motion.

The vector velocity is specified independently of the path to allow continuous motion. The path is specified as a collection of segments. For the purpose of specifying the path, define a special A-B coordinate system whose origin is the starting point of the sequence. Each linear segment is specified by the A-B coordinate of the final point expressed in units of resolution, and each circular arc is defined by the arc radius, the starting angle, and the angular width of the arc. The zero angle corresponds to the positive direction of the A-axis and the CCW direction of rotation is positive. Angles are expressed in degrees, and the resolution is 1/256th of a degree. For example, the path shown in Fig. A-10 is specified by the instructions:

VP	0,10000
CR	10000, 180, -90
VP	20000, 20000

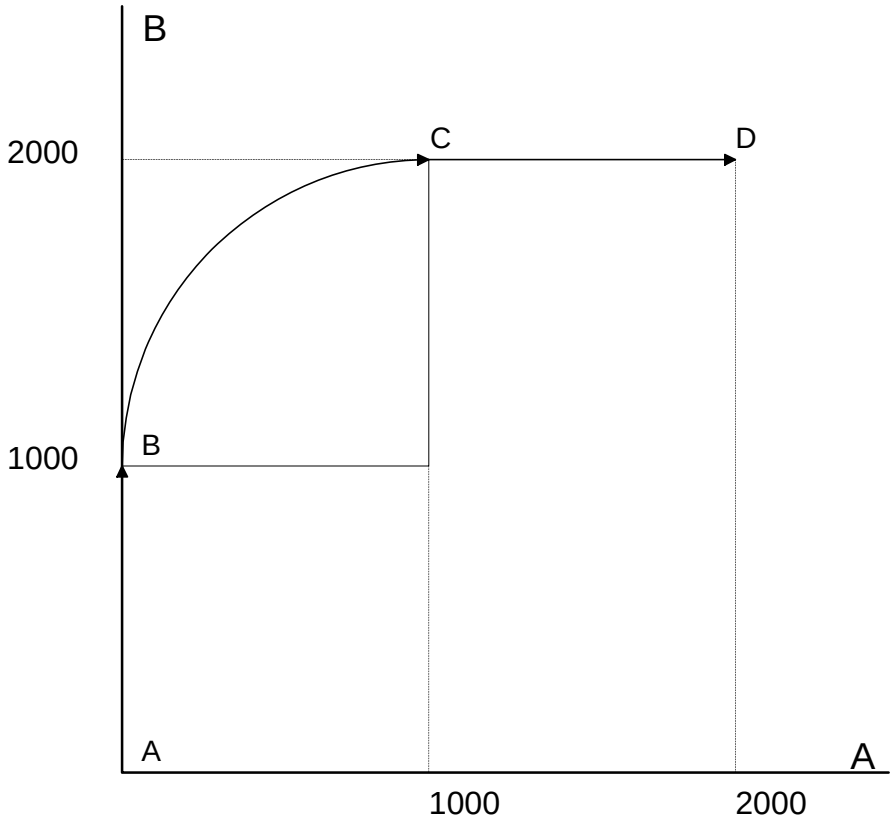


Figure A-10 - X-Y Motion Path

The first line describes the straight line vector segment between points A and B. The next segment is a circular arc, which starts at an angle of 180° and traverses -90°. Finally, the third line describes the linear segment between points C and D. Note that the total length of the motion consists of the segments:

A-B	Linear	10000 units
B-C	Circular	$\frac{R\lvert\Delta\theta\rvert 2\pi}{360} = 15708$
C-D	Linear	10000
	Total	35708 counts

In general, the length of each linear segment is

$$L_k = \sqrt{Xk^2 + Yk^2}$$

Where Xk and Yk are the changes in A and B positions along the linear segment. The length of the circular arc is

$$L_k = R_k\lvert\Delta\Theta_k\rvert 2\pi/360$$

The total travel distance is given by

$$D = \sum_{k=1}^{n} L_k$$

The velocity profile may be specified independently in terms of the vector velocity and acceleration.

For example, the velocity profile corresponding to the path of Fig. A-10 may be specified in terms of the vector speed and acceleration.

VS	100000
VA	2000000

The resulting vector velocity is shown in Fig. A-11.

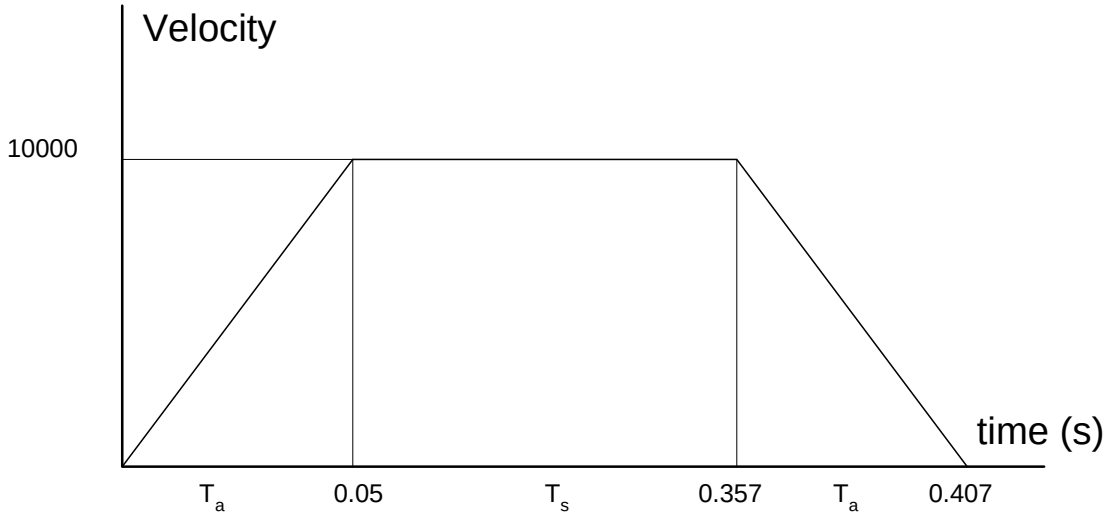


Figure A-11 - Vector Velocity Profile

The acceleration time, T_a, is given by

$$T_a = \frac{VS}{VA} = \frac{100000}{2000000} = 0.05s$$

The slew time, Ts, is given by

$$T_s = \frac{D}{VS} - T_a = \frac{35708}{100000} - 0.05 = 0.307s$$

The total motion time, Tt, is given by

$$T_t = \frac{D}{VS} + T_a = 0.407s$$

The velocities along the A and B axes are such that the direction of motion follows the specified path, yet the vector velocity fits the vector speed and acceleration requirements.

For example, the velocities along the A and B axes for the path shown in Fig. A-10 are given in Fig. A-12.

Fig. A-12(a) shows the vector velocity. It also indicates the position point along the path starting at A and ending at D. Between the points A and B, the motion is along the B axis. Therefore,

$V_b = V_s$

and

$V_a = 0$

Between the points B and C, the velocities vary gradually and finally, between the points C and D, the motion is in the X direction.

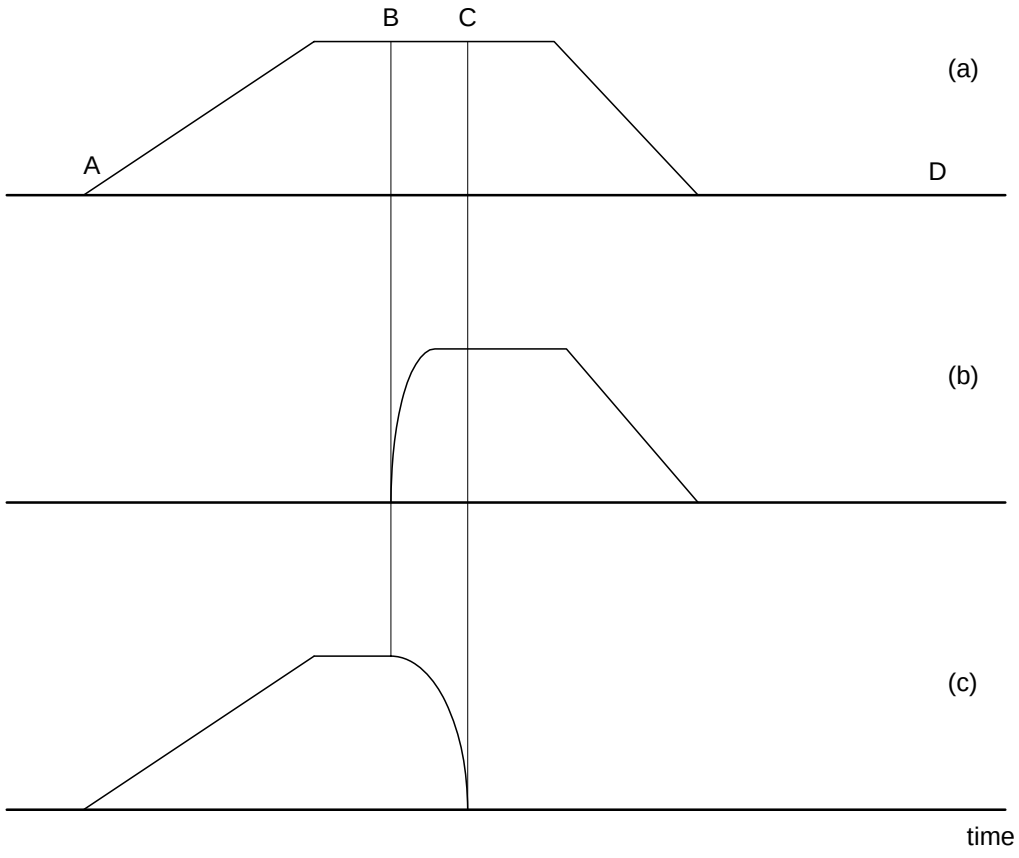


Figure A-12 - Vector and Axes Velocities

List of Other Publications

"Step by Step Design of Motion Control Systems"

by Dr. Jacob Tal

"Motion Control Applications"

by Dr. Jacob Tal

"Motion Control by Microprocessors"

by Dr. Jacob Tal

Training Seminars

Galil, a leader in motion control with over 250,000 controllers working worldwide, has a proud reputation for anticipating and setting the trends in motion control. Galil understands your need to keep abreast with these trends in order to remain resourceful and competitive. Through a series of seminars and workshops held over the past 15 years, Galil has actively shared their market insights in a no-nonsense way for a world of engineers on the move. In fact, over 10,000 engineers have attended Galil seminars. The tradition continues with three different seminars, each designed for your particular skillset-from beginner to the most advanced.

MOTION CONTROL MADE EASY

WHO SHOULD ATTEND

Those who need a basic introduction or refresher on how to successfully implement servo motion control systems.

TIME: 4 hours (8:30 am-12:30 pm)

ADVANCED MOTION CONTROL

WHO SHOULD ATTEND

Those who consider themselves a "servo specialist" and require an in-depth knowledge of motion control systems to ensure outstanding controller performance. Also, prior completion of "Motion Control Made Easy" or equivalent is required. Analysis and design tools as well as several design examples will be provided.

TIME: 8 hours (8:00 am-5:00 pm)

PRODUCT WORKSHOP

WHO SHOULD ATTEND

Current users of Galil motion controllers. Conducted at Galil's headquarters in Rocklin, CA, students will gain detailed understanding about connecting systems elements, system tuning and motion programming. This is a "hands-on" seminar and students can test their application on actual hardware and review it with Galil specialists.

TIME: Two days (8:30 am-5:00 pm)

Contacting Us

Galil Motion Control

3750 Atherton Road

Rocklin, CA 95765

Phone: 916-626-0101

Fax: 916-626-0102

Internet address: support@galilmc.com

URL: www.galilmc.com

WARRANTY

All products manufactured by Galil Motion Control are warranted against defects in materials and workmanship. The warranty period for controller boards is 1 year. The warranty period for all other products is 180 days.

In the event of any defects in materials or workmanship, Galil Motion Control will, at its sole option, repair or replace the defective product covered by this warranty without charge. To obtain warranty service, the defective product must be returned within 30 days of the expiration of the applicable warranty period to Galil Motion Control, properly packaged and with transportation and insurance prepaid. We will reship at our expense only to destinations in the United States.

Any defect in materials or workmanship determined by Galil Motion Control to be attributable to customer alteration, modification, negligence or misuse is not covered by this warranty.

EXCEPT AS SET FORTH ABOVE, GALIL MOTION CONTROL WILL MAKE NO WARRANTIES EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO SUCH PRODUCTS, AND SHALL NOT BE LIABLE OR RESPONSIBLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES.

Index